(1)

HARDWARE IMPLEMENTATION OF A BCH ENCODER, DECODER, AND INTERFACE

THESIS

Norman R. LeClair
Captain, USAF

AFIT/GE/ENG/88D-20

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

(1)

AFIT/GE/ENG/88D-20

HARDWARE IMPLEMENTATION OF A BCH ENCODER, DECODER, AND INTERFACE

THESIS

Norman R. LeClair
Captain, USAF

AFIT/GE/ENG/88D-20

Approved for public release; distribution unlimited.

AFIT/GE/END/88D-20

HARDWARE IMPLEMENTATION OF A BCH ENCODER/DECODER

THESIS

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science in Electrical Engineering

Norman R. LeClair, B.S.

Captain, USAF

December 1988

Preface

This study is a follow-on to the study completed by Capt DeGraff in December 1985. Capt DeGraff constructed a number of circuits to be used to ultimately "calculate the error detection and correction performance of different concatenated coding schemes." Capt DeGraff used a three wire handshake based on a 7474 D flip flop as the serial I/O port for his circuits. This thesis was started with the intent of providing a simulated communications channel which could be used to test the performance of Capt DeGraff's circuits. However, after building the appropriate interface, it was discovered that the 7474 D flip flop would undergo catastrophic failure within 1000 transmitted bits. Therefore, it was necessary to rebuild the BCH encoder and decoder circuits using an Dual Asynchronous Receiver Transmitter (DART) as the I/O port, allowing the rebuilt BCH encoder and decoder to interface to any computer through the computers existing serial I/O ports.

I have found, as all others who have attempted projects of this sort, that I had to rely on the help and support of others. Chief among these is my Lord and Saviour Jesus Christ "For the Lord gives wisdom, and from his mouth comes knowledge and understanding" (Proverbs 2:6). I thank my advisor, Major Glenn Prescott, whose very nature instills

calm reasoning into those around him. Finally a special thanks to my patient and understanding wife, [redacted] "A wife of noble characte who can find? She is worth far more than rubies" (Proverbs 31:10).

Table of Contents

List of Figures

List of Tables

Abstract

A BCH encoder and decoder are implemented in hardware with special emphasis given to the encoder and decoder interfaces. The pitfalls of using a 7474 D flip flop as the basis of building the interface is discussed. The advantages of using a UART for the interface are outlined and the circuit diagrams to implement the UART interface in hardware are provided.

Finite impulse response linear filters are chosen to implement both the BCH encoder and decoder. A basic theoretical understanding of the BCH encoder and decoder function is given. Design decisions made for the hardware implementation of the encoder and decoder are discussed, and schematics detailing the final hardware configurations are provided. Software to run and test all of the above is documented in the appendices.

HARDWARE IMPLEMENTATION OF A BCH ENCODER, DECODER AND INTERFACE

I. Overview

General Issue:

The Foreign Technology Division (FTD) is investigating the performance of error correcting schemes employed in a variety of communications channels. As part of their investigation, FTD has provided AFIT a large thesis topic area which has subsequently been broken into several smaller thesis efforts. The first thesis effort required the construction of a BCH encoder, BCH decoder, interleaver, and deinterleaver in hardware. Captain De Graff GE-85D built the encoders and decoders during the summer of 1985 and documented his designs in his thesis entitled "Hardware Implementation of a Concantenated Encoder/Decoder" (DeGraff, 1985).

The second thesis effort requires the construction of an asynchronous communication channel to provide a computer to BCH encoder/decoder interface. The computer will provide a bit stream to the BCH encoder/decoder as well as the capability for simulating communication channels. This document addresses the second thesis effort.

A third thesis effort will provide the software to

simulate several communications channels. This software, when running on the computer interfaced to the BCH encoder/decoder will provide a test bed to determine the encoders and decoders ability to correct errors.

The final thesis effort will require the construction of a Reed-Solomon encoder decoder scheme in software. The Reed-Solomon encoder/decoder will be interfaced to the communications channel simulator. The final thesis effort ends when data is gathered on the performance of both hardware and software encoder and decoders over a variety of simulated communication channels.

Background:

Communication systems in both commercial and military procurements are moving away from analog technologies in favor of digital technologies. Analog transmissions take on a infinite variety of shapes which must be reconstructed at the receiver for reception of the transmitted information. A small disturbance in an analog system (known as noise) often affects the reception of the received signal, causing distortion (and errors) in the received message.

In contrast to analog systems, digital systems are limited to a finite number of states. When digital transmissions are disturbed by noise, the digital system must chose the closest finite state of the noise-disturbed signal as the received signal. Choosing the closest finite

state often results in the regeneration the original transmitted signal giving digital systems much greater protection against distortion and interference compared to their analog cousins. (Sklar, 1988:3).

Digital communication systems transmit their information in the form of a bit stream. Bit streams can be created many ways, but our discussion is limited to a binary bit stream (composed of two-level pulses, ones and zeros) and transmitted at baseband (unmodulated). The bit stream originates at an information source, referred to as the "source bit stream". For the purposes of this thesis, the source bit stream will be composed of the ASCII set of characters. Each character is defined by 8 bits in a predefined state known as the ASCII bit pattern. Some examples of characters and their associated ASCII bit pattern are given below:

Character	ASCII bit pattern
A	0100 0001
a	0110 0001
L	0100 1100
;	0011 1011
+	0010 1011

Although the transmission of a digital bit stream can be done in a way to minimize the affects of noise, errors in

the transmitted bit stream still occur. Fortunately, encoding the source bit stream prior to transmitting it can provide protection against transmitted errors. Additional bits are added by the encoder to each word of the source bit stream prior to transmission. After reception, the bit stream is passed to the decoder, which uses the additional bits to detect and correct errors if they have occurred during transmission.

Since this thesis centers around the operation of a BCH encoder/decoder, it is worth noting that the BCH encoder multiplies a generator polynomial to a character (or source word) to create the additional bits used for error detection and correction (more about this in chapter four). The character plus the additional bits are known collectively as the code word. The code word, when received by the decoder, allows detection and correction from 1 to several errors depending on the nature of the generator polynomial chosen.

Problem Statement:

Determine the feasibility of interfacing a Z-100 computer to an existing BCH encoder/decoder based on a 7474 D flip flop serial I/O port. Make all modifications necessary to the BCH encoder/decoder to produce a working final product.

Scope:

A BCH encoder/decoder was designed and constructed by Capt Peter de Graff for his thesis during the summer of 1985. Capt de Graff's design utilized a 7474 D flip flop to provide a three wire handshake interface. The interface includes a data ready line, data request line, and the data line itself. The interface is controlled by a Z-80 CPU driven by software located in a 2719 E-Prom.

This thesis will determine the feasibility of interfacing directly to the 7474 D flip flop of the BCH encoder from a Z-100 computer. The interface from the Z-100 to the 7474 D flip flop will be controlled from the bus structure of the Z-100. Design heuristics will be employed to modify both the Z-100 interface and the BCH interface as necessary to provide fully functional asynchronous communications between the computer and the BCH encoder.

Approach:

A hardwire interface from the Z-100 to the BCH encoder will be constructed on a wire-wrap board. The initial design of the hardwire interface will use a 7474 D flip flop to provide compatibility with the BCH encoder/decoder designs. The interface will be mounted on an extender card which in turn is plugged into the S-100 bus of the Z-100 computer. Control over the interface will be accomplished by the Z-100 CPU issuing control signals over the S-100 bus at a specified port address. Isolation of the bus from

spurious signals will be accomplished by appropriate data and address buffers.

During construction of the initial interface, test software will be written to help verify the design process. Each stage of the design will be independently tested, along with the final completed design. Once the interface is completed it will be tested for fully asynchronous communications. All problems encountered with the initial design will be documented.

If the initial design fails to provide adequate asynchronous communications, a design modification to the BCH encoder will be chosen that will provide fully asynchronous communications. The design modification, if needed, will be implemented in hardware and tested for proper operation.

Materials and Equipment:

1. Z-100 computer
2. Assorted IC's
3. Logic Analyzer
4. Logic Probe
5. Wire-wrap board
6. Voltage regulators with heat sinks
7. Assorted tools
8. Ribbon cables with connectors
9. MASM software

10. Two Z-80 DARTs

11. Function generator (156 Khz square wave)

12. Multimeter

13. Oscilloscope

Summary:

A prototype BCH encoder will be interfaced to a Z-100 computer to provide the encoder with a source bit stream. A prototype BCH decoder will also be interface to a Z-100 computer to provide the decoder with a received bit stream and to allow the simulation a communication channel. Both system will be tested to insure fully functional asynchronous serial transmission can be sustained by the encoder's and decoder's I/O port. The procedure for demonstrating the final product can be found in chapter six.

II. 7474 D Flip Flop Interface

Introduction:

A BCH encoder/decoder, and an interleaver/deinterleaver were designed and built by Capt DeGraff in a prior thesis effort. Capt DeGraff's design used a 7474 D flip flop as the basis for serial interface between the components of his system. These interfaces were to allow a source bit stream to supply a source word to the encoder, the encoder to supply the encoded source word (code word) to either a channel simulator or a deinterleaver (in the concatenated mode), the channel simulator to supply the transmitted code word (corrupted with noise) to the decoder (or deinterleaver in the concatenated mode), and finally the decoder to supply the decoded word to a bit sink. The source bit stream, channel simulator, and bit sink were all to be supplied as software routines run on dedicated computers. The final system configuration is shown below:

Each interface was designed to support a three wire handshake for data transmissions. The handshaking includes a data line, data ready line, and data request lines. The received bit stream is clocked directly into the system component and can not contain any parity, checksum or stop bits. Since no direct interface exists from any Z-100 I/O

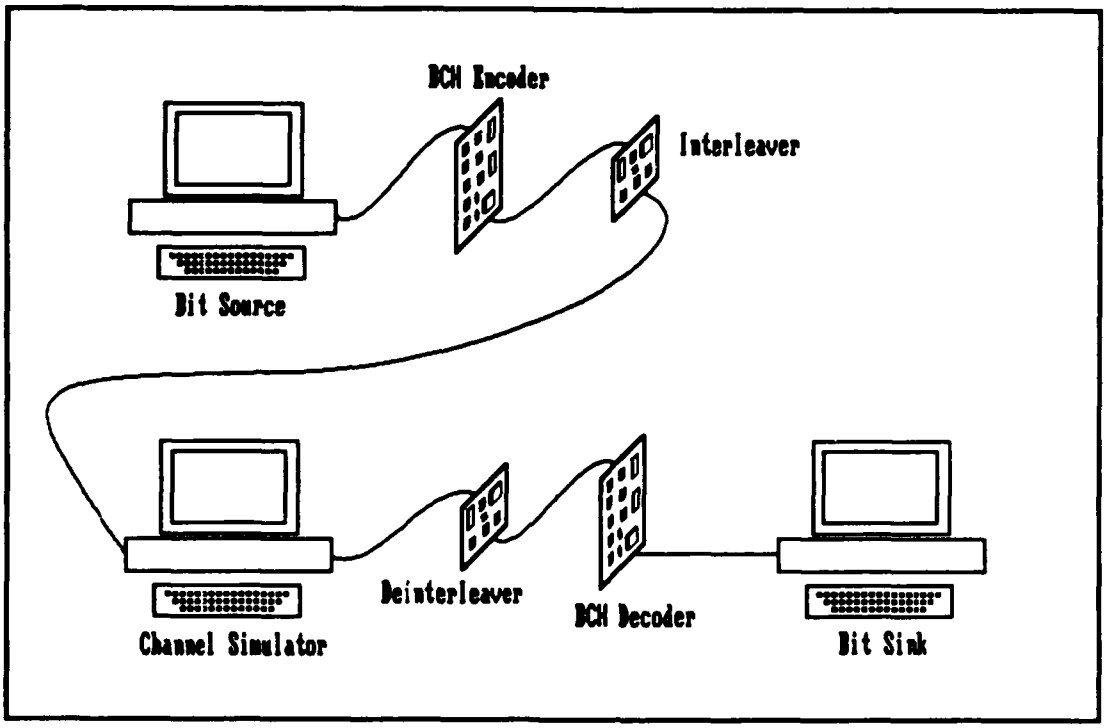

Figure 1 - System Configuration

port to the BCH encoder, a special serial I/O port from the Z-100 to the BCH encoder must be designed to support data transmissions (DeGraff, 1987:8).

Background:

To better appreciate the design problem presented above, it is best to spend some time gaining an understanding of the interface designed into the BCH encoder. The interface is based on a 7474 D flip flop and is shown in the block diagram below:

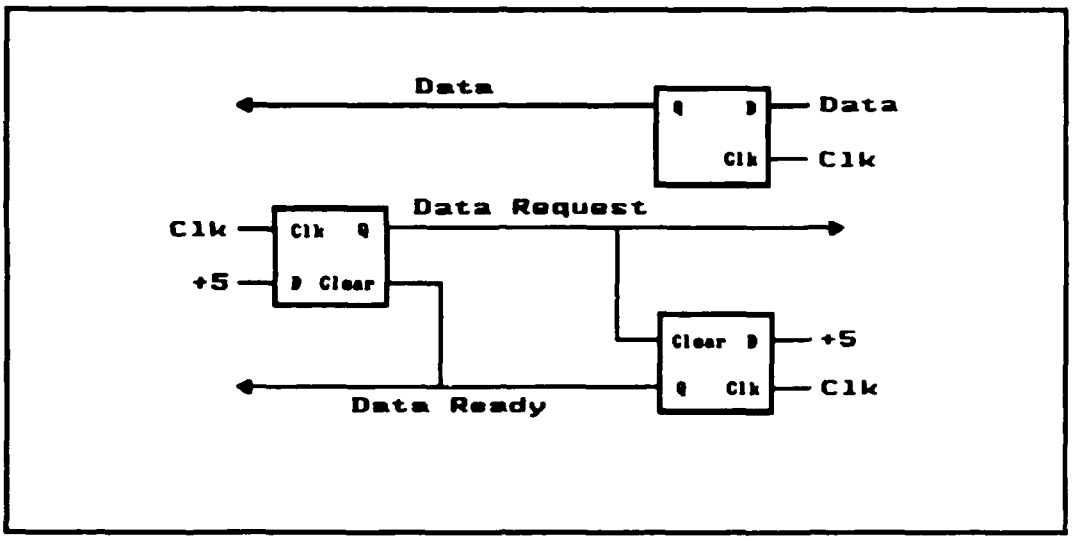

Figure 2 - 7474 D Flip Flop Interface

The sequence of events that accomplish data transmissions between the two 7474 D flip flops are as follows:

<u>One:</u> The flip flop requesting data sets its data request line high. This automatically clears the sending flip flop's data ready line. The receiving flip flop starts polling the data ready line to see if data has been place on the data line.

<u>Two:</u> During this time the sending flip flop polls the data request line to see if the receiving flip flop wants some data. When the request line goes high, the sending flip flops places one bit of data on the data line, and sets

its data ready line high. Setting the data ready line high automatically resets the data request line on the receiving flip flop. Once the data request line is reset, the sending flip flop is free to resume its polling of the data request line.

Three: When the data ready line goes high, the receiving flip flop clocks the data into its processor. As soon as the processor is finished processing the bit, the receiving flip flop sets the data request line high to request another data bit. Setting the data request line high automatically sets the data ready line low on the sending flip flop and the process repeats again.

As can be seen in the diagram, two 7474 D flip flops are needed to interface any two pieces of equipment for one way (half duplex) data transmissions. For two half duplex transmissions a total of four 7474 D flip flops are needed, two flip flops for half duplex signal. Both the encoder and decoder boards contained two 7474 D flip flops enabling them to perform two half duplex transmissions. Typically, the encoder will receive a bit stream from some source (a Z-100 computer in our case), encode the received bit stream and send the encoded bit stream to a transmitter. Since the received bit stream and the encoded bit stream interface with different devices (a source and a transmitter) it is

expedient to have two separate half duplex serial ports on the encoder. Similarly, the BCH decoder needs to interface to a receiver and to a bit sink such as a CRT that can display a message.

Design Decisions:

Design decisions made prior to implementing an interface between the Z-100 computer and the BCH encoder/decoder are as follows.

Z-100 Serial Ports: Neither the serial A port nor the serial B port of a Z-100 can be used to support data transmissions to or from the BCH encoder. Both ports are implemented using a Motorola 2661 universal asynchronous receiver transmitter. While the 2661 IC is ideally suited to asynchronous serial transmissions, it cannot be program to exclude either start or stop bits. Since neither the BCH encoder or decoder circuitry can tolerate stop and start bits, the serial A and serial B ports cannot be used for the interface.

Z-100 Parallel Port: The parallel data port on the Z-100 might possibly support data transmission from the Z-100 to either the BCH encoder or decoder. However, no MACRO function MASM (Micro Assembler) call exits to the operation shell that allows data to be imported through the parallel

I/O port. In fact, even if a MACRO call could be constructed that would allow inputting into the parallel port, there is only one parallel port available on each computer when two would be needed to implement a channel simulator. Therefore the serial I/O port was ruled out as a means of accomplishing the interface.

7474 D Flip Flop: The interface currently existing on the BCH encoder and decoder utilizes a 7474 D flip flop to implement a three line handshake between the encoder and decoder for the transmission of data. Fortunately, it is possible to construct an I/O port based on a 7474 D flip flop that will interface to the S-100 bus of a Z-100 computer. Each computer is configured with up to three expansion slots on the S-100 bus making interfacing to the bus very easy. Once the flip flop is interfaced to the bus, the Z-100 can effectively control all operations of its flip flop exactly as the BCH encoder or decoder controls operations of its flip flop. Therefore, the method of choice for constructing an I/O port from the Z-100 to the BCH encoder is to use a 7474 D flip flop based on the design inherent within the BCH encoder and decoder.

Design Parameters:

The design of the Z-100 serial data interface based on a 7474 D flip flop centered around three design parameters:

1) control by the Z-100 over the interface, 2) characteristics of the 7474 D flip flop, and 3) buffering of address and data lines. Each of these design parameters are addressed below:

Control Functions: The implementation of effective data communications to and from any I/O port will ultimately allow the host computer complete control over the I/O port. A Z-100 computer normally uses assembly language function calls to exercise control over its existing serial I/O ports. These calls are IN and OUT instructions embedded within a software routine than transfers data to and from the port. Both the IN and OUT instructions use a specified port address to direct the flow of data. Data is clocked into or out of the port by special processor signals that occur during the execution of the IN of OUT instruction. These processor signals are available on the S-100 bus of the Z-100 computer.

To implement control by the Z-100 computer over the 7474 D flip flop both IN and OUT instructions are used along with their associated processor signals. In order to specify a unique port address for the IN and OUT instructions, an address decoder was construction for both the input port and the output port (see Figures 3 and 4). Address bit A1 is inverted for the output port, but not for the input port.

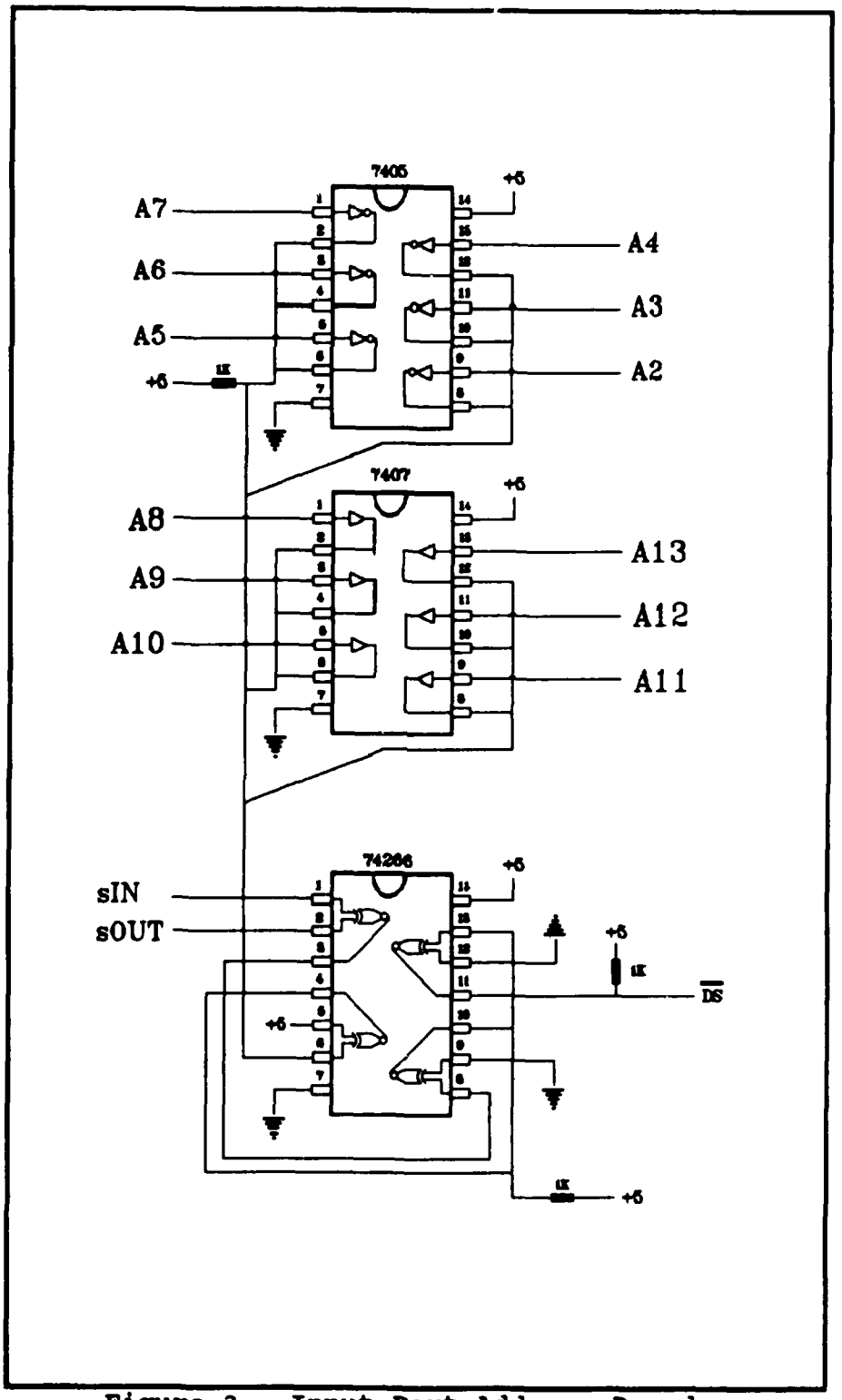

Figure 3 - Input Port Address Decoder

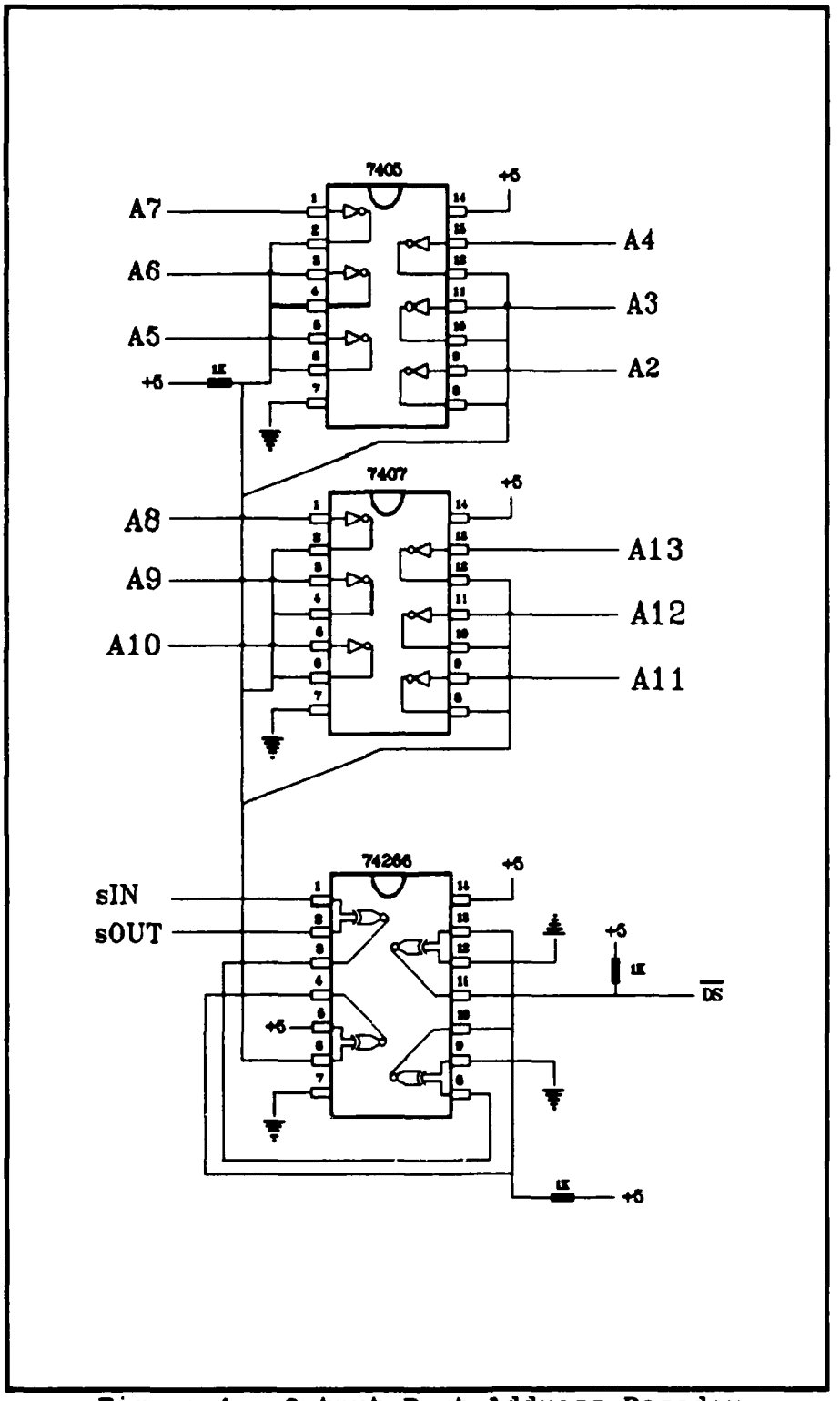

Figure 4 - Output Port Address Decoder

7474 D Flip Flop Characteristics: The 7474 D flipflop consist of two delay or following flip flops on one integrated circuit. Each flip flop comes with is own preset, clear, clock, and Q output lines. The clock line on a 7474 D flip flop is extremely sensitive and must be isolated by a buffer to prevent inadvertent triggering of the clock by noise. Note that the data request line is tied to the clk of D2 on the output port flip flop. This data request line cannot be read as a status bit (data request high) without triggering the clock. Therefore, the data request line is split by a 7407 hex non-inverting buffer into two separate lines. One line isolated is to the clock alone, the other line is used to read in the status bit (data request).

Buffering of Lines: All data lines are buffered by a DP8216 data buffer to isolate the S-100 data lines from those used by the I/O ports. Data lines leading into and out of the ports are active only when the port address is active. All address lines leading into the address decoder are isolated by either 7405 or 7407 hex buffers to prevent address lines from inadvertently being drawn either high or low by the port operations.

Signal Description:

Control of the input and output ports by the Z-100 CPU

requires the Z-100 to provide signals to clear and set the data ready line, clear and set the data the request line, and clock in data both to and from the data line. Both the input and output port control signals are discussed separately below:

Input Port:

Data Request Line: The data request line is enabled (high) by presetting Q1 of the 7474 D flip flop. The preset input to the 7474 D flip flop is active low. A low signal is provided by the CPU to the flip flop when A0, sOUT, and DS are all concurrently high. A0, sOUT, and DS will all be high when the CPU issues an OUT command at address 03h (note that DS* has been conditioned by an invertor to yield DS).

Data Ready Line: The data request line is disabled (low) when the data request line is enabled. Disabling the data request line is most easily accomplished by letting Q2 of the 7474 D flip flop represent the state of the data ready line. Then, tying the clear input of Q2 on the 7474 D flip flop to the same line controlling the data request line insures that Q2 will always be cleared when the data request line is preset. When the data ready line goes high, Q2 is clocked into its high state by the incoming data ready signal, telling the CPU that data on the data line is

ready to be clocked into the computers memory.

Data: Data being clocked into the input port is isolated by a 7407 non inverting buffer as it enters the input port. Leaving the input port, the data is buffered by a DP8216 quad line buffer. The DP8216 is enabled by DS* and controlled by pDBIN. Therefore, data can only be clocked into memory when the CPU issues an IN command at either address 02h or 03h.

Clear Data Request Line: During the initialization of the input port, it is helpful to clear the data request line. The data request line can be cleared by enabling the clear input of the 7474 D flip flop to Q1. A0, sIN, and DS are combined by a triple input nand gate to provide the control signal needed. When the CPU issues an IN instruction at address 03h, the data request line will be cleared.

Output Port:

Data Request Line: The data request signal originates at an input port and is viewed as an incoming signal by the output port. The incoming data request signal is monitored by the CPU so that the CPU knows when to place new data on the data line. The data request line also sets the data ready line of the output port low. The data ready

line is represented by Q2 of the output port's 7474 D flip flop. The incoming data request line is tied to the clock of the 7474 D flip flop. Since D2 is tied to ground, only a low signal can be clocked into Q2. Hence, when the data request line goes high, the data ready line will go low.

Data Ready Line: The data ready line is place high by the CPU whenever new data is place on the data line. Q2 of the output port's 7474 D flip flop is used to represent the data ready signal. Q2 can be enabled by enabling the present input (active low) to the flip flop. DS, pDBIN, and sOUT are combined by a triple input nand gate to provide the control signal needed. When the CPU issues a OUT command at address 00, the data ready line will go high.

Data: Data is represented by Q1 of the output ports 7474 D flip flop. Data is clocked into Q1 when the data ready line goes high. This is accomplished by tying the clock input of the flip flop to the same control signal that sets the data ready line high. The D input to the flip flop is tied to DO0 of the S-100 bus through a DP8216 quad line buffer. Data is clock into the flip flop whenever the CPU issues the command OUT at address 00.

Clear Data Ready Line: During initialization of the output port, it is helpful to clear the data ready line.

The data ready line can easily be cleared by enabling the clear input to Q2 of the flip flop. A0, sIN, and DS are combined by a triple input nand gate to provide the control signal. When the CPU issues an IN command at address 01, the data ready line is cleared.

Circuit Diagrams: Circuit diagrams, including pin connections, are provided in Figure 5 (input port) and Figure 6 (output port) on the following two pages.

Test Methodology:

Introduction: The test methodology was developed to provide a quantifiable measure of the 7474 D flip flop interface's performance. The interface design was deemed acceptable when it could receive a 10 megabit source bit stream (in 8 bit blocks) into its input port and transmit it back to the source through its output port without error. The source bit stream was generated from a test bed that could transmit and receive data based on a three wire handshake protocol.

Test Bed: Capt DeGraff's BCH encoder was originally considered for use as the system interface test bed. Unfortunately, the encoder IC is not easily accessible to

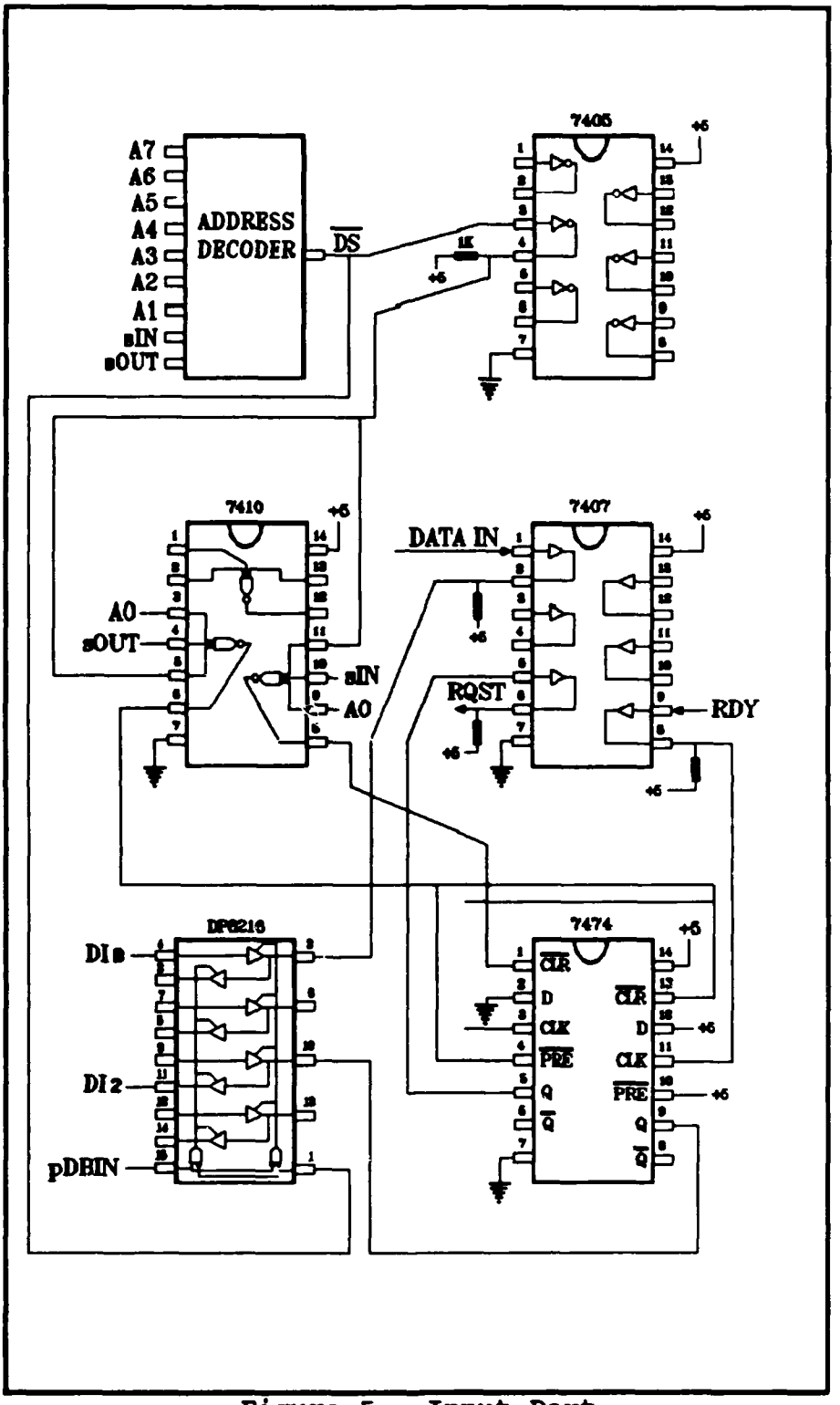

Figure 5 - Input Port

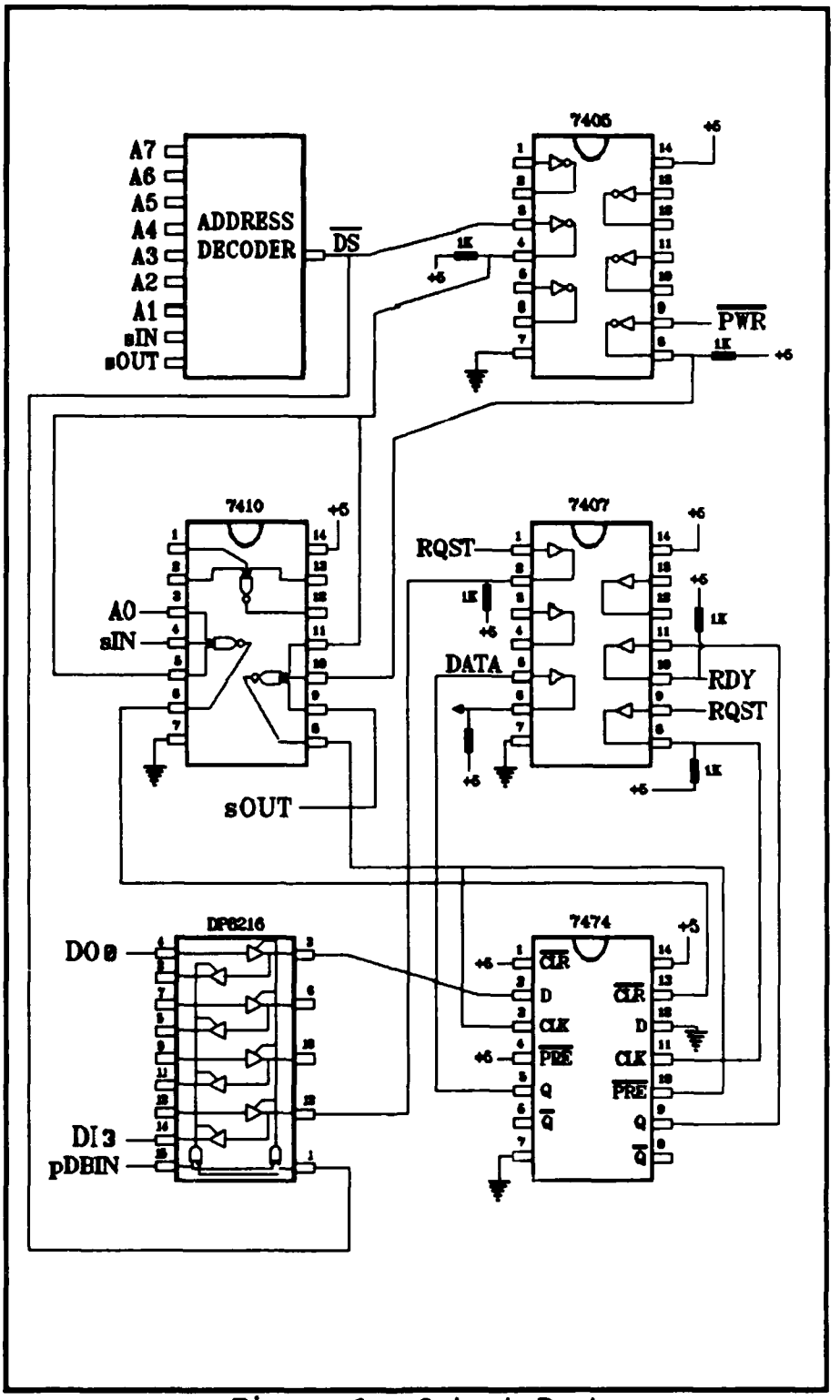

Figure 6 - Output Port

probes making it difficult to trap and analyze the data received by the encoder. In addition, the BCH encoder has never been tested at high bit rates (4800 baud and above) it is not known whether the BCH encoder has any design flaws that might interfere with the serial transmissions at high bit rates. Therefore, the BCH encoder was ruled out for use as test bed to test the 7474 D flip flop interface.

An identical 7474 D flip flop interface running on a second Z-100 provides a better alternative test bed. Each interface consists of one input and one output port both of which have identical designs (based on the interface existing on the BCH encoder). Since the designs are analogous to the BCH encoder, testing the interface using a second Z-100 will yield the same results as testing the interface using the BCH encoder. In addition, all data and control lines on the Z-100 are easily accessible, making the trapping and analysis of data and signal flow relatively easy.

Test Methodology: The test methodology chosen is incremental in nature and consists of two steps. Each step requires a software driver to be written. All software drivers are included in appendix A.

Step One: The input port and the output port of the Z-100 interface are tied together on a single computer.

The ports are exercised for the proper transmission and reception of data. Two separate software drivers are used:

Driver One: Accept a character from the keyboard and transmit it out the output port to the input port. Receive the character from the input port and display it on the screen. A visual check of the screen will show whether the transmission is handled correctly.

Driver Two: Take a single character and transmit it out the output loop to the input port. Receive the character from the input port and check the character internally to determine if the receive character matches the sent character. Report the number of characters sent and the number of errors found on the CRT screen. Repeat the process indefinitely.

Step Two: The input port of the first Z-100 is tied to the output port of the second Z-100 and the input port of the second Z-100 is tied to the output port of the first Z-100. A software driver is written for each Z-100:

First Z-100: Accept a character from the keyboard, echo it to the screen and transmit the character out the output port to the second Z-100. Watch the input port to see if the data ready line goes high. When it does,

receive the character from the input port and echo the character to the screen. It should be the same character as the one originally sent. Repeat this process indefinitely.

Second Z-100: Receive a character from the input port, echo it to the screen and output this character to the output port. Repeat the process indefinitely.

Test Results:

Testing of the interface on a single Z-100 (step one) resulted in the successful transmission of over 10 Mbits without error. The second test (step two) had some surprising results. The transmission invariably failed within 1000 transmitted bits. The failure was catastrophic, with no ability to recover transmission. Types of failures occurring included missed bits (the failure of the receiving Z-100 to receive all 8 bits), improper bits (bits that are inverted in state), and failure to transmit because both Z-100's fall into a receive only mode.

Discussion:

Failure to transmit between the two separate Z-100s probably occurred because the lines were sampled as the lines were changing state (amounting to a contention between states). Unfortunately, it is impossible to program the existing lab equipment to trap the status of all signals

when such a contention arises, so it is impossible to prove the actual cause of the failure.

However, the failures observed are well known and documented for networks, and the solution to network contentions has traditionally been the addition of protocol to the transmission. Since the two Z-100s, when attempting to communication to one another through the customized I/O ports, constitute a crude network, it seems reasonable to add protocol to their transmissions to solve the observed contention problems.

The addition of protocol to the software drivers was in fact tried. Appendix B contains the software used. The protocol attempted to determine if the character (8 bits) was sent in its entirety, and if not, the character was retransmitted. In addition, the protocol attempted to determine if a single bit was transmitted correctly by repeatedly sampling the transmitted bit. Although the protocol was successful in raising the average number of bits transmitted before a failure occurred, it did not succeed in preventing failures.

The question arises as to why transmission on a single Z-100 (to and from) were so successful (10 Mbits without an error) when transmissions between two Z-100s were so inadequate. This can be explained by the execution of the software driver written to handle the data transmission. The software takes one bit of a character, places in on the

output port, then goes to the input port to see if the bit is there. Since it takes the execution time of several instructions for the Z-100 to switch from outputting the bit to determining if it is at the input port, the 7474 D flip flop has had ample time to complete its transitions and stabilize. The net results is that when the Z-100 goes to look for the bit at the input port, the bit is always there and always stable. The software acts as a type of synchronization between transmission. Of course, this type of synchronization will never occur between two separate Z-100s (with separate internal clocks) trying to talk to each other.

More protocol could be added (beyond what was initially tried) by adding start bits, stop bits, and parity. Unfortunately, the BCH encoder, as designed will not support such additions of protocol. The BCH encoder must receive raw data alone, the encoder has no provisions to strip out any support bits.

Conclusion:

An interface between a Z-100 and a BCH encoder based on a 7474 D flip flop is unworkable. A new interface that supports asynchronous serial transmissions is needed.

Recommendation:

An interface base on a Universal Asynchronous Receiver

Transmitter (UART) should be added to the BCH encoder. A UART provides RS232 standard serial transmissions that can be tied directly to any computers serial I/O ports (no customized ports are needed). The UART provides protocol in the form of start, stop, and parity bits. In addition, the UART can be programmed to synchronize its sampling of transmissions based on a start bit, virtually eliminating the chance of missampling a bit.

III. Universal Asynchronous Receiver Transmitter Interface

Introduction:

The previous chapter concluded with the statement that the serial I/O port to the BCH encoder is unworkable as it is presently designed. The encoding function of the BCH encoder is implemented by a finite impulse response (FIR) filter and can be easily isolated from the serial I/O port function. Modifying the BCH I/O port based on a Universal Asynchronous Receiver Transmitter (UART) will completely restore the BCH encoders functionality, and by using a Dual Asynchronous Receiver Transmitter (DART), the BCH encoder will have the two separate I/O ports it needs to meet its interface requirements.

Background:

The DART is composed of two Universal Asynchronous Receiver Transmitters (UARTs). Each UART provides the capability for half duplex transmissions over a single serial port along with an interface for modem control. Control lines to the UART allow the designer complete control over UART functions.

The DART relieves the user from the task of breaking a data byte into single bits, transmitting the bits serially, and

reassembling the bits. The user simply loads the DART with the byte to be transmitted and receives reassembled bytes from the DART upon the return transmission. All start bits, stop bits, and parity bits are added by the DART independently of user action. Data and framing errors are relayed to the user through a special status register located in the DART.

Data lines into and out of the UART are triply buffered to insure data byte isolation. The UART is programmable from 300 to 800K bits per second, and provides internal sampling synchronization based on the start bit and the programmed transmission rate. The internal sampling synchronization can be program from 1 to 32 times the bit rate. By selecting a sampling synchronization rate of 16, for example, the DART automatically sets its internal sampling clock to 16 times the transmission rate. The DART will sample the incoming bit stream every 16 clock pulses (of the internal clock), starting its first sample at the center of the pulse representing the start bit. The DART effectively synchronizes its sampling of the incoming data to the data transmission rate thereby preventing the sampling of data during the data's transition from one state to another (Campbell, 1987: 151-161).

Design Decisions:

The DART will be configured to interface directly to the serial port of a Z-100 computer. Using the J2 port (modem or Data Terminal Equipment (DTE) port) requires the DART to be configured as a Data Communication Equipment (DCE) device. The DART will be configured to transmit at 9600 baud to interface to the Z-100 modem port. Experience has shown that for short transmissions, no parity bit is required. Therefore the parity bit will be disabled on both the DART and Z-100 ports.

Using the Z-100 serial port limits the design to strict RS-232 standards. Therefore the BCH encoder will use a DP1488 Quad Line Driver and DP1489 Quad Line Receiver to condition the transmission line signals to RS-232 standards (± 12 volts, see Figure 7).

The BCH encoder will undergo a prototype design on a S-100 wirewrap board. Using the S-100 wirewrap board enables the design to interface directly with a Z-100 CPU through the S-100 bus located on the Z-100 mother board. All control signals that are software programed will be provided to the BCH encoder by the Z-100 CPU through the S-100 bus. Designing the BCH encoder on the S-100 wirewrap board allows the designer greater flexibility in monitoring signals and implementing design changes. All data lines between the DART and the Z-100 are buffered by DP8216 bidirectional bus transceivers. A customized address decoder is

provided to isolate chip enable control signals to a specific command address (see Figure 7).

Signal Description:

The DART is controlled by the CPU through signals provided on the S-100 bus. Details of these control signals are provided in the following discussion (note that the pin numbers given indicate pins found on the DART):

1. Pins 5, 6, & 7 (INT*, IEI, IEO). Both interrupt enable signals (IEI and IEO) are tied to ground. Interrupts are not used to implement the design. Since interrupts are disabled, INT* will not affect the DART operation. INT*, therefore, is allowed to float.

2. Pin 8 (M1*). M1*, machine cycle one, is used in conjunction with IORQ* to acknowledge an interrupt. Since interrupts are disabled, M1* is disabled by tying it to +5 volts.

3. Pin 9 (V_{DD}). Tied to +5 volts.

4. Pin 10 (W/RDYA*). Normally, the wait/ready states are user defined to one function (either wait or ready). Once defined, W/RDYA* is used to implement I/O control from the CPU to the Z-80

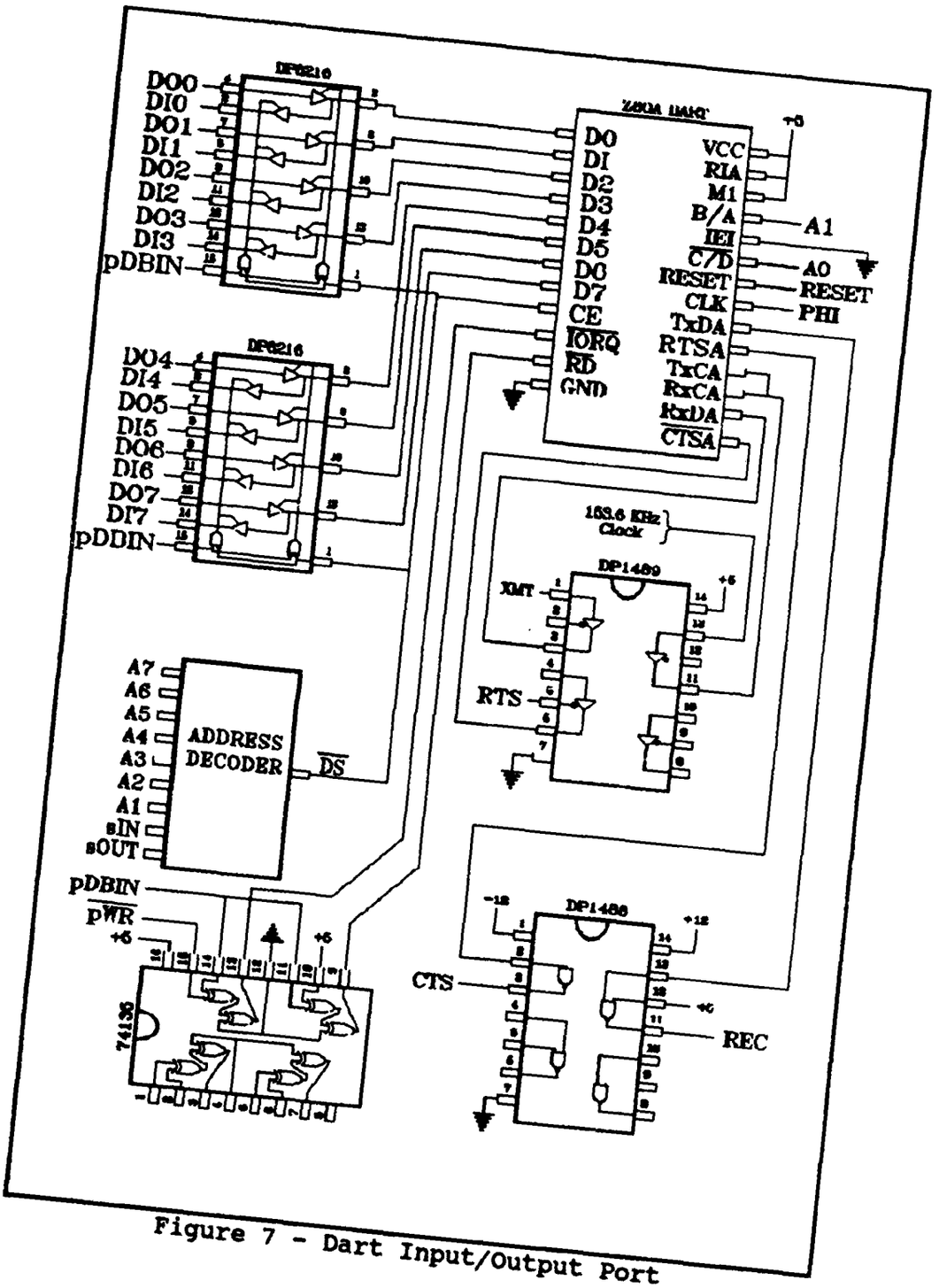

Figure 7 - Dart Input/Output Port

Dart and vice versa. The BCH encoder design will utilize software to implement DART control by polling the appropriate Z-80 DART status registers and providing appropriate I/O functions based on the returned status bits. W/RDYA* is not used in the design and is therefore allowed to float.

5. Pin 11 (R/A*). R/A*, ring indicator for channel A, is used to receive a request for connection in switched line operations. When the input goes low, the Z-80 DART interrupts the CPU. This function is not implemented in the design. Therefore, R/A* is disabled by tying it to +5v.

6. Pin 12 (RxDA). Data is received by the Z-80 Dart at this pin from the serial bit stream. No control is implemented here, however, the line into pin 12 is conditioned by the DS1489 Quad Line Receiver.

7. Pins 13 and 14 (RxCA* and TxCA*). The receiver and transmitter clocks are programmed for 9600 baud. The design implements a divide by 16 command for the clock inputs. This requires 153.6 KHz at the inputs of pins 13 and 14.

8. Pin 15 (TxDA). TxDA is used to transmit data from the Z-80 DART to a receiving DTE device. TxDA requires no control

functions. The outgoing line is, however, conditioned by the DS1488 Quad Line Driver.

9. Pin 16 (DTRA*). DTRA*, data terminal ready, is an output signal used to inform the receiver of a ready condition. It is not used in the design and is therefore allowed to float.

10. Pin 17 (RTSA*). RTSA*, ready to send, is shown connected to pin 5 (CTS - clear to send) of the DTE interface. When RTSA is enable, the receiving transmitter is enabled if the DTE device is in auto enable mode. This tells the DTE device to send the data that is loaded into its transmitter buffer. The line from the Z-80 to the H-29 is conditioned by a DS1488 Quad Line Driver to meet RS-232 standards.

11. Pin 18 (CTSA*). CTSA*, clear to send, is connected to pin 4 (RTS - request to send) of the H-29 DTE interface. The DART is not configured for auto enable, therefore CTSA* cannot not enable the DART's transmitter. Instead, the CPU will poll the status register which will indicate that CTSA has been enabled by the DTE device. When the CPU knows that CTSA is enabled, the CPU will load the transmitter buffer with a data byte. The transmitter of the DART is always in the enabled mode and the loaded byte will automatically be sent. Since CTSA* must meet

RS-232 standards, it is conditioned by a DS1489 Quad Line Receiver.

12. Pin 19 (DCDA*). Data carrier detect (DCDA*) is clocked into register 0, bit 3, of the Z-80 Dart. Our design shows DCDA* connected to pin 20 (DTR - data terminal ready) of the DTE interface to provide terminal on/off condition information to the Z-80 DART. Similar information is available through CTSA*, that is, the H-29 terminal must be on to provide a request to send signal.

13. Pin 20 (CLK). Tied directly to the CPU clock - Phi.

14. Pin 21 (RESET*). Tied directly to the CPU reset - RESET*.

15. Pin 22 through 30. These pins are used for channel B and do not apply to the design. They are allowed to float.

16. Pin 31 (Gnd). Tied directly to system ground.

17. Pin 32 (RD*). RD*, read, is an input line used to inform the Z-80 DART when the CPU is reading data from memory or from an I/O. This is equivalent to the CPU state that occurs when pDBIN goes high. Therefore pDBIN is connect through an invertor to RD*

of the Z-80 DART.

18. Pin 33 (C/D*). C/D*, control or data select input line provides the CPU either control or data input conditions for the Z-80 DART. When this line is a logical 0, the D0-D7 pins on the Z-80 DART are used for data I/O. When the line is a logical 1, the D0-D7 pins are used for control information. The system address line A0 provides this information to the DART. Address 00h corresponds to channel A - I/O data port, address 01h corresponds to channel A - Status and Command port. Note that the least significant bit, A0, provides the necessary condition for the selection of control or data input as needed. Therefore, A0 is tied directly to pin 33.

19. Pin 34 (B/A*). Although channel B is not used for the initial BCH encoder design, provisions have been made to expand the design to include channel B. The design allows a future expansion to include channel B by tying pin 34 to address line A1. Then all address to either 00 or 01 will access channel A while all addresses to either 02 or 03 will access channel B.

20. Pin 35 (CE*). Chip enable is tied directly to DS* of the address decoder. When either address 00h or 01h for channel A or 02h or 03h for channel B is used by the CPU, the Z-80 DART is

enabled. See figure 3 for a schematic of the address decoder.

21. Pin 36 (IORQ*). IORQ*, input and output request, is an input signal to the Z-80 DART. It is used by the CPU to inform the Z-80 DART that the CPU is performing an input or an output operation. Our design implements this control signal through the use of pDBIN and pWR*. A truth table showing input signal states and the desired output is given below along with a logic diagram implementing the truth table criteria.

pDBIN	pWR*	Desired Output	Exclusive Or
0	0	0	0
0	1	1	1
1	0	N/A	1
1	1	0	0

Note: pDBIN should never be high when pWR* is low. This equates to a read and write cycle occurring simultaneously. Our design disable IORQ* under this condition to complete the truth table.

Figure 8 shows the control signal implementation for both IORQ* and RD*. Exclusive Or gates were chosen for both circuits to minimize chip count.

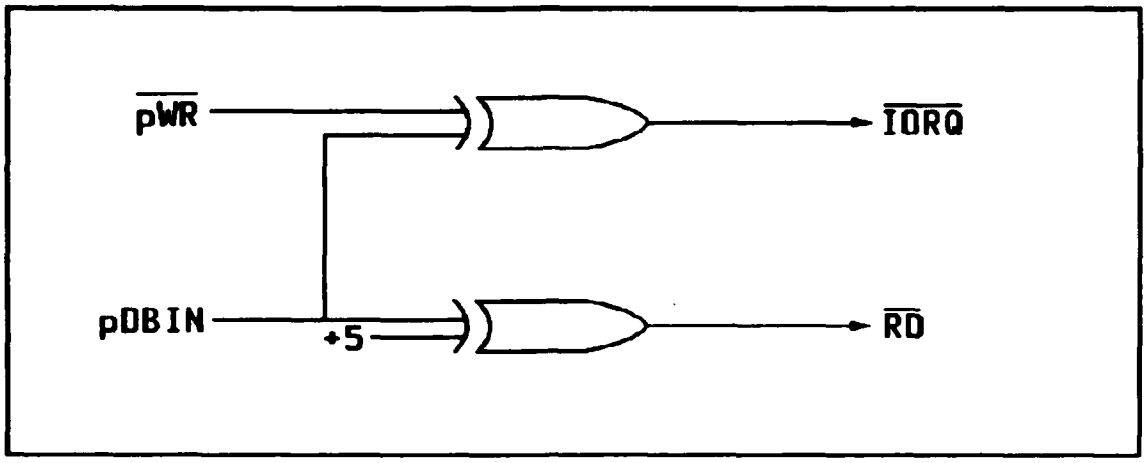

Figure 8 - RD* and IORQ* Implementation

Test Methodology:

To meet operational requirements, the DART must be able to transmit or receive at least 10M bits of information, consecutively, without error. Two software drivers were written to implement the test (see appendix C).

Driver One: From a Z-100, initialize the modem port (J2) to 9600 baud, one stop bit, no parity, 8 bits/character, and x16 clock. Transmit one character (8 bits) out the J2 port directly to the DART located on a second Z-100. Once the character has been transmitted, poll J2 for a received character (the DART will retransmit the character back to J2). Extract the received character from J2 and compare it to the sent character. Repeat the process for 10M bits while reporting the outcome (number of

characters sent and number of bad characters received) to the CRT.

Driver Two: Using a Z-100 to send commands to a DART through the S-100 bus, initialize the DART to 9600 baud, one stop bit, no parity, 8 bits/character, and x16 clock. Receive a character into the DART, echo the character to the CRT, and retransmit the character back to the first Z-100.

Test Results:

Five test runs were conducted over two days. Each test run required about 2 hour and 15 minutes to transmit 10.4 Mbits at 9600 baud from a Z-100 to the DART and back to the Z-100. Each word (8 bits) of information transmitted was compared to the received word to determine the number of errors occurring. The first test run was started while the equipment was at room temperature. Data collected during the test showed errors occurring at the rate of one error per word transmitted. The error rate drop to zero as the equipment warmed up to the operating temperature. Data from the first test can be found in appendix D.

The four remaining test runs were all conducted at the system operating temperature. No errors were recorded for the first three test runs. The fifth test run recorded two errors,

both occurring within the last 5 minutes of the test.

A final test run was conducted one week later. For this test, the external clock was set at 166.6 KHz (6 usec/clock pulse). At 166.6 KHz, the system performed well both at room temperature and at operating temperature.

Discussion and Conclusions:

Errors in transmissions occur because the sampling of a particular bit is erroneous. The sampling rate for the DART is tied to an external clock, which was originally set at 153.6 KHz. A change in the external clock rate will result in sampling errors when the change takes the external clock outside of its proper operating range.

When the clock was set at 153.6 KHz, it is possible that the clock was set just outside of its proper operating range, causing errors to occur at room temperature. As the clock warmed up, it shifted to a slightly higher frequency, just inside its normal operating range, allowing the DART to work perfectly at operating temperatures.

The two errors of the fifth test run may be attributed to random noise. They represent a error rate of 1 in 20.8 M bits transmitted and can be tolerated for the purposes of this thesis.

A DART configured for one stop bit, one start bit, no parity, 9600 baud, and x16 clock will work best with the external

clock set at 166.6 KHz.

IV. BCH Encoder

Introduction:

As mentioned earlier in this thesis, FTD has an interest in testing the performance of a BCH encoder over several communication channels. Chapter four describes the BCH encoder, the circuitry used to build the encoder, and the methodology used to test the encoder for correct operation.

Background:

The BCH code originated from the work of three individuals (Bose, Chaudhuri, and Hocquenghem) in the early 1960s. The first letter of each individuals last name was combined to name the code (BCH). The BCH codes are cyclic in nature, mean that rotating a code word any number of bits either left or right will still result in a code word. BCH codes are often thought of as a generalization of the Hamming code and are used as single or multiple error correcting codes. (Lin and Costello, 1983:141)

A discussion of codes, encoding, and decoding begins with the length of the code word (n), the length of the source word to be encoded (k), and the number of errors to be corrected in any given code word (t). Encoding is the process of taking a source word of length k and adding error checking bits to the word resulting in a code word of length

n. The idea is to add enough error checking bits to the code word so that single or multiple errors can be found and corrected after the code word has been transmitted and received. Describing a particular coding scheme is done by giving the length of the code word and the length of the source word in parenthetical notation (n,k) followed by the number of errors the coding scheme can correct. As an example, one useful coding scheme is describe as a (15,7) double error correcting BCH code.

BCH codes are characterized by the following parameters (given m is an integer ≥ 3 and $t \leq 2^{(m-1)}$):

1) $N = 2^m - 1$.

2) Number of error checking digits $\leq m * t$.
Note that the number of error checking digits also equals n - k.

(Blahut, 1984:162)

The BCH code characteristics given above result in the following table of BCH codes for $m \leq 6$:

Table 1 - BCH Codes for $m \leq 6$:			
m	n	k	t
3	7	4	1
4	15	11	1
4	15	7	2
4	15	5	3
5	31	26	1
5	31	21	2
5	31	16	3
5	31	11	5
5	31	6	7
6	63	57	1
6	63	51	2
6	63	45	3
6	63	39	4
6	63	36	5
6	63	30	6
6	63	24	7
6	63	18	10
6	63	16	11
6	63	10	13
6	63	7	15

To generate a BCH code word, the encoder receives a source word and multiplies it by a generator polynomial. For the purpose of this thesis, the source word, generator polynomial, and the code word are all represented as binary polynomials. For instance, the source word 11 (in bits) is represented as $x + 1$. The source word 1001 is represented as $x^3 + 1$. The multiplication of the source polynomial and the generator polynomial is strictly algebraic but follows the rules for exclusive-or binary addition to combine terms.

Calculating the generator polynomial requires the use

of prime polynomials of degree m within a Galois Field (2^m) and the determination of the minimal polynomial for i = i .. 2t over the source words within the Galois Field. For a more through discussion of Galois Fields and minimal polynomials, the reader is referred to Blahut pages 162-166.

Generator Polynomial:

Transmitting bits of information between the Z-100 and the BCH encoder is easiest to accomplish if the number of bits transmitted is the length of a byte (8 bits) since the MASM calls and registers within the Z-100 are built to handle a byte of information at a time. Each byte transmitted is taken from the ASCII character set and is 7 bits in length. The eighth bit transmitted is always defined, but not really needed to represent the ASCII character. Therefore, a BCH code that has k equal to 7 or 8 bits is most useful for the transmissions that are occurring between the Z-100 and the BCH encoder.

For $m \leq 6$, the BCH codes with k = 7 or 8 are:

(15,7) 2 error correcting BCH code

(63,7) 15 error correcting BCH code

To calculate the generator polynomial for the (15,7) BCH code (m=4), a Galois Field of $GF(2^4)$ is chosen using the primitive polynomial $p(z) = z^4 + z + 1$. The minimal polynomials for all the field elements over $GF(2^4)$ is given

in Table 2:

Table 2 - Minimal Polynomials over $GF(2^4)$	
Field Element(s)	Minimal Polynomial
a^1, a^2, a^4, a^8	$x^4 + x + 1$
a^3, a^6, a^9, a^{12}	$x^4 + x^3 + x^2 + x + 1$
a^5, a^{10}	$x^2 + x + 1$
$a^7, a^{11}, a^{13}, a^{14}$	$x^4 + x^3 + 1$

(Blahut, pg 163)

The generator polynomial is found by multiplying the Lowest Common Multiple (LCM) of the minimal polynomials for the field elements ranging from a^1 .. a^{2t} where t equals the number of errors corrected by the BCH code. For the (15,7) 2 error correcting code chosen, the generator polynomial (g(x)) is given as:

$$g(x) = \text{LCM}\ (a^1, a^2, a^3, a^4)$$

$$g(x) = (x^4 + x + 1)(x^4 + x^3 + x^2 + x + 1)$$

$$g(x) = x^8 + x^7 + x^6 + x^4 + 1$$

or in binary:

$$g(x) = 111010001$$

(Blahut, 1985:163-164)

Multiplier Circuitry:

Fortunately, many simple circuits taken from the family of finite impulse response (FIR) (non-recursive) digital filters work beautifully to multiply polynomials. The BCH

encoder constructed for this thesis uses a linear-feedforward shift register FIR (shown below) for its multiplication.

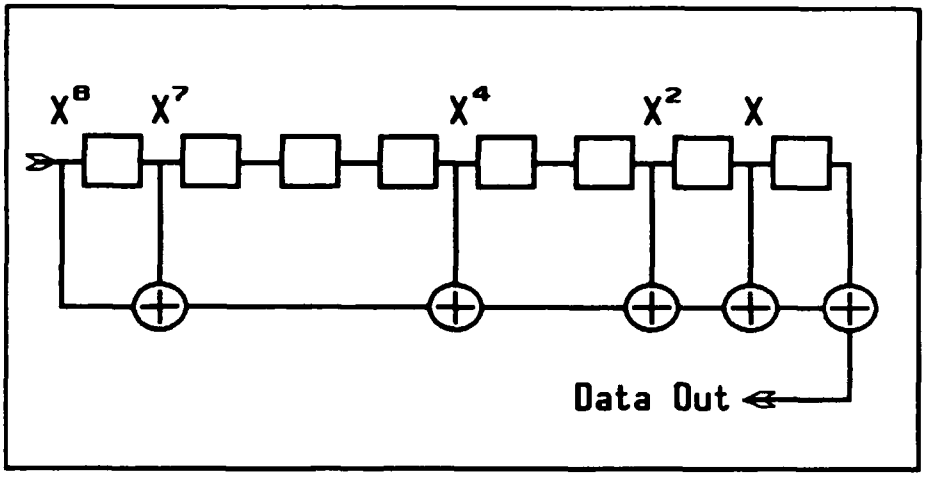

Figure 9 - FIR Multiplication Circuit

A quick example demonstrates the operation of the FIR. The generator polynomial shown can be represented by $x^8 + x^7 + x^4 + x^2 + x + 1$. The generator polynomial will be multiplied by the code word 01100001 (ASCII "a"). Note that the high order bit of the ASCII "a" enters the encoder first and that the high order bit of the encoded word (the result) is read from the FIR first. Each stage of the multiplication is given in Table 3.

Table 3 - Register States for Multiplication										
Clock	R1	R2	R3	R4	R5	R6	R7	R8	R9	Result
1	0	0	0	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	1
3	1	1	0	0	0	0	0	0	0	0
4	0	1	1	0	0	0	0	0	0	1
5	0	0	1	1	0	0	0	0	0	0
6	0	0	0	1	1	0	0	0	0	1
7	0	0	0	0	1	1	0	0	0	1
8	1	0	0	0	0	1	1	0	0	0
9	0	1	0	0	0	0	1	1	0	1
10	0	0	1	0	0	0	0	1	1	0
11	0	0	0	1	0	0	0	0	1	1
12	0	0	0	0	1	0	0	0	0	1
13	0	0	0	0	0	1	0	0	0	0
14	0	0	0	0	0	0	1	0	0	1
15	0	0	0	0	0	0	0	1	0	1
16	0	0	0	0	0	0	0	0	1	1
17	0	0	0	0	0	0	0	0	0	1

The result is 0101011010110111 (binary), or $x^{14} + x^{12} + x^{10} + x^9 + x^7 + x^5 + x^4 + x^2 + x + 1$.

Control Circuitry:

The BCH encoder is controlled by three signals: one to clear the encoder, one to clock data into the encoder, and one to clock data out of the encoder. Each control signal must be isolated from all other control signals to the BCH encoder as well as control signals occurring on the S-100 bus and control signs to the Z-80 DART. Isolation of the control signals is accomplished through a 74139 Dual 2 to 4 Decoder. The decoder enable input is tied to DS* linking the decoder function directly to the Z-80 DART and isolating the decoder from any S-100 internal control functions.

Pins A1 and B1 of the decoder are tied to address lines A14 and A15, neither of which are used by the Z-80 DART's address decoder. A14 and A15 provide control over the decoder operation independently of Z-80 DART operation. Since the 74139 control lines (Y1 through Y3) are asserted (low) ONLY when A14 and/or A15 are asserted (high), the BCH encoder is active only when address 4000, 8000, and c000 (hex) are asserted. DART operations must be disabled (Tx disabled) because the DART will misinterpret OUT commands at the BCH encoder address as an instruction to load its transmit buffer.

Data is routed into the DART through DO7 of the S-100 bus and is not affected by DART operations. Data is routed from the BCH encoder into the Z-100 through DI0. DI0 is multiplexed from both the BCH encoder and Z-80 DART to insure that input data is always taken from one source at a time.

The encoder clock and encoder clear functions are narrowed by combining pWR* on the S-100 bus with the appropriate 74139 decoder output lines. The narrowed control pulses are necessary to prevent inadvertent triggering of the 74139 output lines when DS* changes during transitions of A14 and A15. These state changes can cause Y2 of the 74139 to momentarily go low, clearing the BCH encoder registers which (obviously) prevents the multiplication of the code word with the generator

polynomial.

Schematics:

Circuit diagrams are given for both the BCH encoder and the control circuitry in Figures 10 and 11 on the following pages.

Test Methodology:

The BCH encoder must successfully encoded a given set of code words for a variety of generator polynomials to be deemed fully acceptable. The test methodology chosen requires the BCH encoder to encode the ASCII set of characters (the code words) using generator polynomials varying from order 1 to order 8. Beyond order 8, it becomes impractical for the Z-100 register set to handle the polynomial multiplication.

A single software test routine was written to test the BCH encoder (see appendix E). The test routine prompts the user for the generator polynomial as an ASCII character - 30 hex. Subtracting 30(hex) from an ASCII character results in numerics (1-9) being represented as their binary value in the Z-100 registers. The generator polynomial entered by the user must be set on the BCH encoder using the appropriate dip switches.

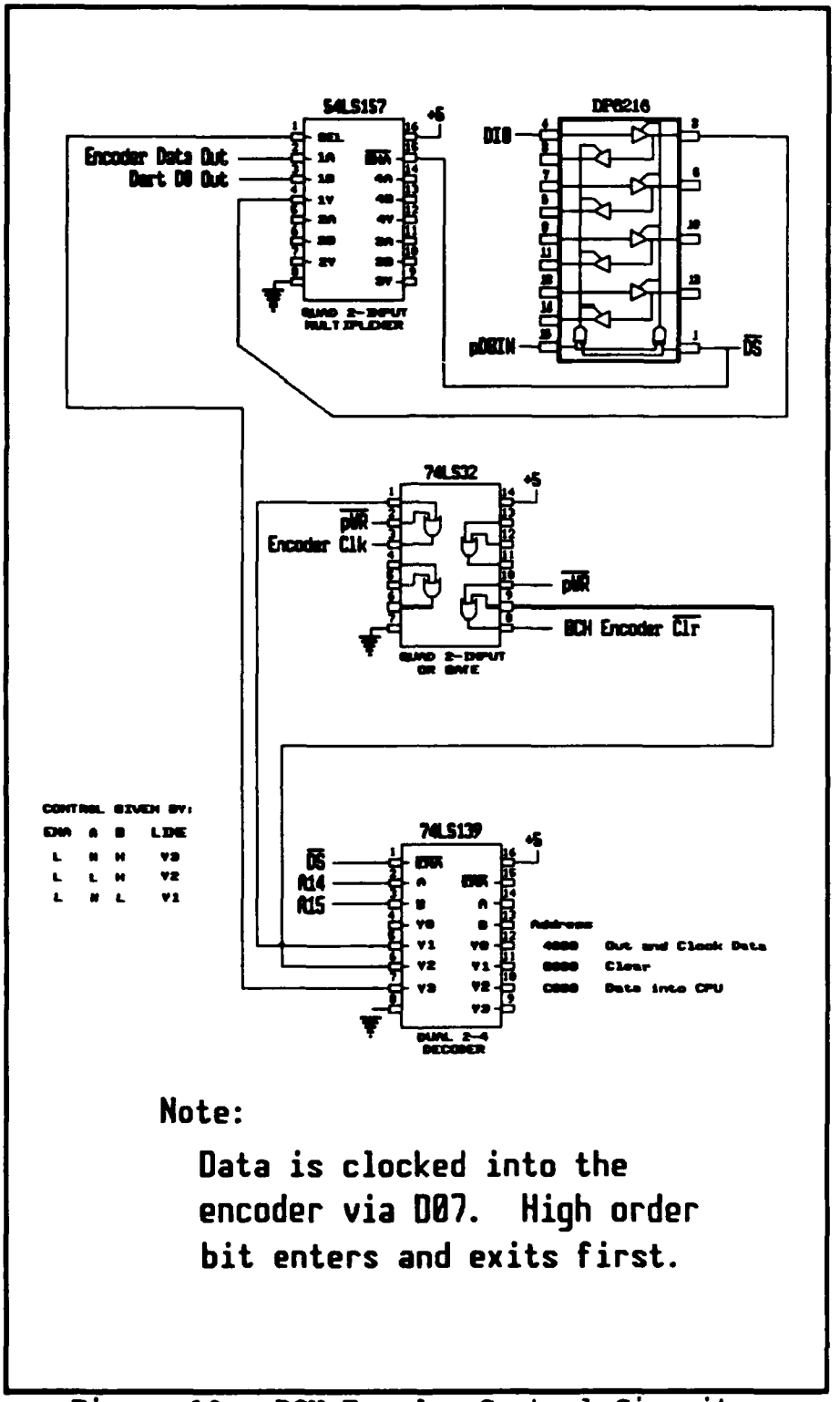

Figure 10 - BCH Encoder Control Circuitry

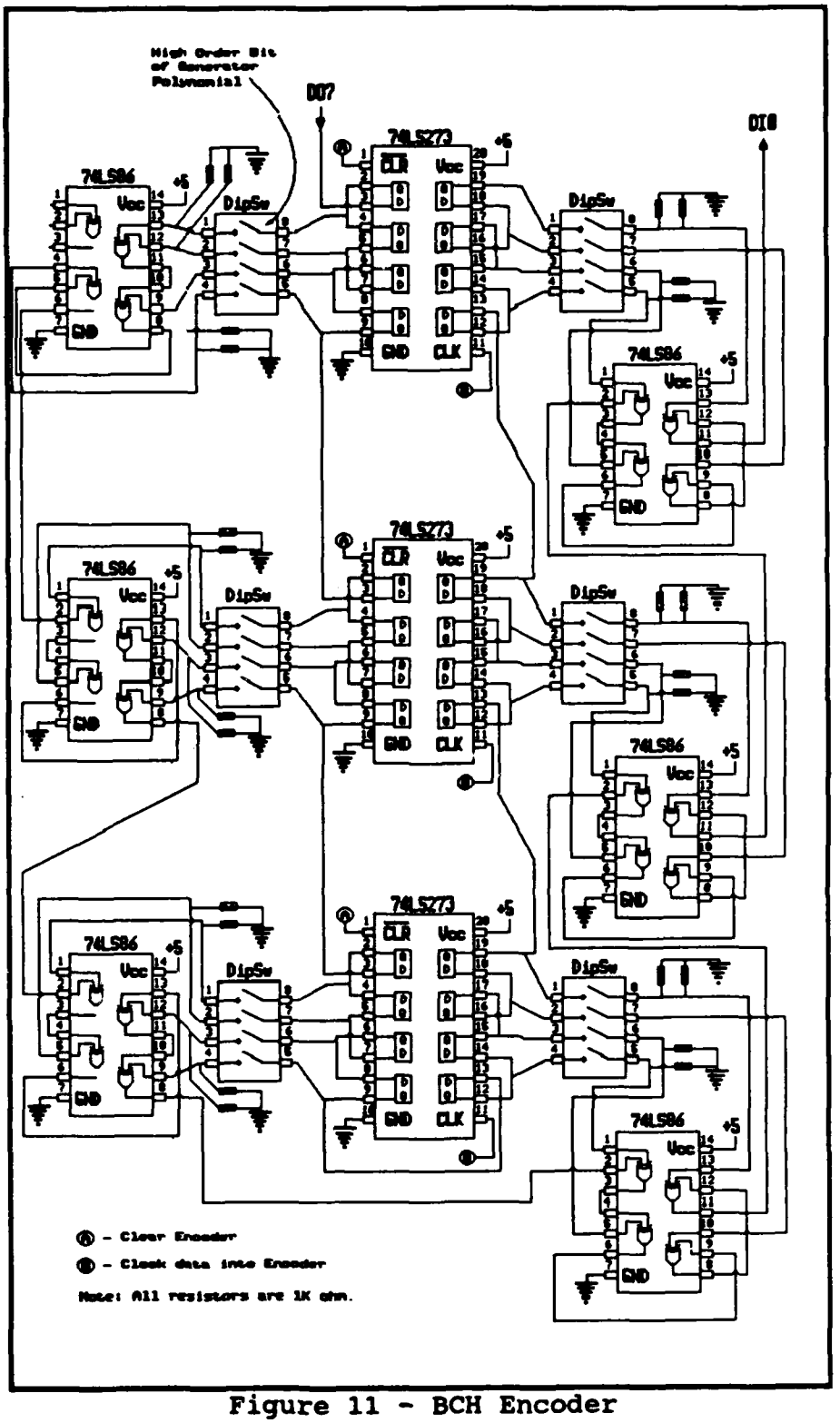

Figure 11 - BCH Encoder

Once the generator polynomial has been entered into the computer by the user, the computer will prompt the user for the value of n-k (or the order of the generator polynomial). With this information the computer will format its registers to conduct the polynomial multiplication.

Next, the computer prompts the user for a ASCII character (the source word). After the source word is entered, the computer generates the code word using a software routine and displays the code word on the screen. Then the computer sends the source word through the BCH encoder, padding the source word with the appropriate number of zeros. The code word received from the BCH encoder is displayed on the screen directly under the code word generated by the computer. If the two code words are identical, the BCH encoder worked correctly for that source word.

Test Results:

Three generator polynomials (1001, 10001, and 111010001 - binary) were selected to test the BCH encoder. Source words were picked at random from the keyboard. In all cases the encoder worked perfectly.

Conclusions and Recommendations:

The linear feed-forward shift register chosen to implement the BCH encoder performs extremely well. The BCH

encoders performance remains stable at the data rates needed to interface to a Z-100 computer.

V BCH Decoder

Introduction:

The counterpart to the BCH encoder is the BCH decoder. The decoder receives the code word after the code word has been transmitted over free space from the encoder. The decoder is responsible for correcting any errors in the code word (up to a specified limit) that might have occurred during transmission, and decoding the corrected code word to produce the original transmitted source word. Chapter five describes the nature of a BCH decoder, the circuitry used to build a BCH decoder, and the test methodology used to test the final product.

Background:

A BCH code word is generated by selecting a source word and multiplying the source word by a given generator polynomial to produce the code word. Decoding the code word uses the same generator polynomial to divide the code word, resulting in the original source word as the quotient of the division.

A problem arises, however, when the transmitted code word is received with an error in it. The error in the code word can cause a different quotient (and hence a

different source word) to be produced by the decoder. Fortunately, the decoder can detect when the code word is erroneous by looking at the remainder of the division process. If no error is present when the code word is received, the remainder will always be zero. If an error is present the remainder will always be greater than zero. The remainder of the division process is known as the syndrome and is the length of the order of the generator polynomial (n-k). The only time the decoder will fail to detect an error in the received code word is when the errors introduced by the communications channel into the transmitted code word produce a different, but correctly encoded, source word.

Calculating the Syndrome:

For cyclic codes (such as the BCH code), the syndrome is determined by:

$$S(x) = R_{g(x)}[V(x)]$$

where:

$S(x)$ = the code word

$R_{g(x)}$ = the remainder after dividing by $g(x)$

$V(x)$ = the received code word

Note, however, that the received code word can be written as the transmitted code word exclusive-ored to an

error polynomial (exclusive-or denoted as a + sign):

$$S(x) = R_{g(x)}[e(x) + C(x)]$$

$$S(x) = R_{g(x)}[e(x)] + R_{g(x)}[C(x)]$$

$$S(x) = R_{g(x)}[e(x)] + 0$$

(1) $$S(x) = R_{g(x)}[e(x)]$$

where:

$e(x)$ = error polynomial

$C(x)$ = transmitted code word

(Blahut, 1984:99)

Equation 1 shows that all the syndromes for a cyclic code can be found by dividing all possible error polynomials by the generator polynomial. Each resulting quotient after the division becomes the syndrome for the particular error polynomial.

For example, choosing the (15,7) two error correcting BCH code with a generator polynomial of $x^8 + x^7 + x^6 + x^4 + x + 1$ yields the following table of syndromes for the given error polynomials:

Table 4 - Syndromes for a (15,7) BCH Code

Error Polynomial	Syndrome Polynomial
0000000000000001	0000000000000001
0000000000000010	0000000000000010
0000000000000100	0000000000000100
0000000000001000	0000000000001000
0000000000010000	0000000000010000
0000000000100000	0000000000100000
0000000001000000	0000000001000000
0000000010000000	0000000010000000
0000000100000000	0000000011010001
0000001000000000	0000000001110011
0000010000000000	0000000011100110
0000100000000000	0000000000011101
0001000000000000	0000000000111010
0010000000000000	0000000001110100
0000000000000011	0000000000000011
0000000000000101	0000000000000101
0000000000001001	0000000000001001
0000000000010001	0000000000010001
0000000000100001	0000000000100001
0000000001000001	0000000001000001
0000000010000001	0000000010000001
0000000100000001	0000000011010000
0000001000000001	0000000001110010
0000010000000001	0000000011100111
0000100000000001	0000000000011100
0001000000000001	0000000000111011
0010000000000001	0000000001110101
0100000000000001	0000000011101001
0000000000000110	0000000000000110
0000000000001010	0000000000001010
0000000000010010	0000000000010010
0000000000100010	0000000000100010
0000000001000010	0000000001000010
0000000010000010	0000000010000010
0000000100000010	0000000011010011
0000001000000010	0000000001110001
0000010000000010	0000000011100100
0000100000000010	0000000000011111
0001000000000010	0000000000111000
0010000000000010	0000000001110110
0100000000000010	0000000011101010
0000000000001100	0000000000001100

Table 4 Continued
Syndromes for a (15,7) BCH Code

Error Polynomial	Syndrome Polynomial
0000000000010100	0000000000010100
0000000000100100	0000000000100100
0000000001000100	0000000001000100
0000000010000100	0000000010000100
0000000100000100	0000000011010101
0000001000000100	0000000001110111
0000010000000100	0000000011100010
0000100000000100	0000000000011001
0001000000000100	0000000000111110
0010000000000100	0000000001110000
0100000000000100	0000000011101100
0000000000011000	0000000000011000
0000000000101000	0000000000101000
0000000001001000	0000000001001000
0000000010001000	0000000010001000
0000000100001000	0000000011011001
0000001000001000	0000000001111011
0000010000001000	0000000011101110
0000100000001000	0000000000010101
0001000000001000	0000000000110010
0010000000001000	0000000001111100
0100000000001000	0000000011100000
0000000000110000	0000000000110000
0000000001010000	0000000001010000
0000000010010000	0000000010010000
0000000100010000	0000000011000001
0000001000010000	0000000001100011
0000010000010000	0000000011110110
0000100000010000	0000000000001101
0001000000010000	0000000000101010
0010000000010000	0000000001100100
0100000000010000	0000000011111000
0000000001100000	0000000001100000
0000000010100000	0000000010100000
0000000100100000	0000000011110001
0000001000100000	0000000001010011
0000010000100000	0000000011000110
0000100000100000	0000000000111101
0001000000100000	0000000000011010
0010000000100000	0000000001010100
0100000000100000	0000000011001000
0000000011000000	0000000011000000

Table 4 Continued
Syndromes for a (15,7) BCH Code

Error Polynomial	Syndrome Polynomial
0000000101000000	0000000010010001
0000001001000000	0000000000110011
0000010001000000	0000000010100110
0000100001000000	0000000001011101
0001000001000000	0000000001111010
0010000001000000	0000000000110100
0100000001000000	0000000010101000
0000000110000000	0000000001010001
0000001010000000	0000000011110011
0000010010000000	0000000001100110
0000100010000000	0000000010011101
0001000010000000	0000000010111010
0010000010000000	0000000011110100
0100000010000000	0000000001101000
0000001100000000	0000000010100010
0000010100000000	0000000000110111
0000100100000000	0000000011001100
0001000100000000	0000000011101011
0010000100000000	0000000010100101
0100000100000000	0000000000111001
0000011000000000	0000000010010101
0000101000000000	0000000001101110
0001001000000000	0000000001001001
0010001000000000	0000000000000111
0100001000000000	0000000010011011
0000110000000000	0000000011111011
0001010000000000	0000000011011100
0010010000000000	0000000010010010
0100010000000000	0000000000001110
0001100000000000	0000000000100111
0010100000000000	0000000001101001
0100100000000000	0000000011110101
0011000000000000	0000000001001110
0101000000000000	0000000011010010
0110000000000000	0000000010011100

It should become clear, from the exhaustive example in the table above, that for a known BCH code all given error

polynomials have a unique syndrome. The BCH decoder can determine the error pattern of the received code word by matching the syndrome of the received code word to the syndromes corresponding error polynomial. Once the error polynomial is known, the received code word can be corrected by exclusive-oring the error polynomial to the received code word. The decoder will take the corrected received code word and decode it to yield the original source word and a syndrome composed solely of zeros.

D. DECODER CIRCUITRY

The BCH decoder can be easily implemented using simple circuits from the family of finite impulse response (FIR) non-recursive digital filters. For the purpose of this thesis, an internal Xor circuit was choosen to ease the implementation of the feedback gate. The internal Xor circuit requires one additional chip to form the feedback gate while the external Xor circuit requires enough chips to form a gate at each point of feedback in the generator polynomial. An example of an internal xor FIR circuit, using $x^8 + x^7 + x^6 + x^4 + x + 1$ as the generator polynomial is given in figure 12.

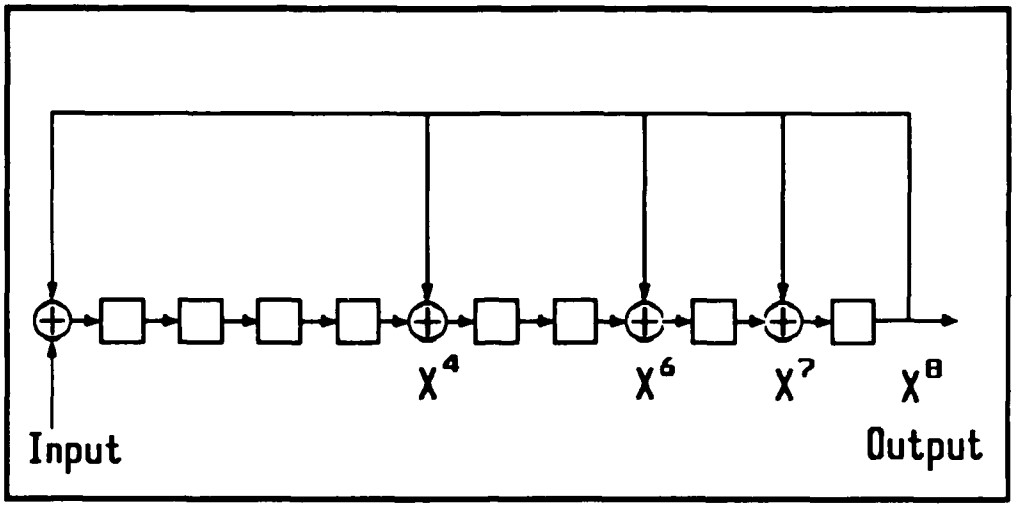

Figure 12 - FIR Dividing Circuit

An example of the decoder operation is given below. For the example, an ASCII "a" is encoded to yield 100111110110001 (binary) as the input polynomial. The generator polynomial (both for encoding and decoding) is given as $x^8 + x^7 + x^6 + x^4 + x + 1$. The first 8 bits of the code word are clocked into the decoder with the feedback gate closed. The next seven bits are also clocked in the decoder with the feedback gate close, but each resulting bit is saved by the CPU to form the decoded word. When the last bit of the result is stored in the CPU, the syndrome (remainder) of the division is residing in the registers of the decoder. The syndrome is clocked into the CPU by opening the feedback gate, and entering zeros into the input of the decoder. A total of 8 bits must be clocked out of

the decoder to form the complete syndrome. Note that high order bits enter the decoder first and leave the decoder first.

Table 5 - Register States for Division

Input	R1	R2	R3	R4	R5	R6	R7	R8	Output
1	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0
1	1	1	1	0	0	1	0	0	0
1	1	1	1	1	0	0	1	0	0
0	1	1	1	1	1	0	0	1	1
1	1	1	1	1	0	1	1	1	1
1	0	1	1	1	0	0	0	0	0
0	1	0	1	1	1	0	0	0	0
0	0	1	0	1	1	1	0	0	0
0	0	0	1	0	1	1	1	0	0
1	0	0	0	1	0	1	1	1	1
0	0	0	0	0	0	0	0	0	0

Control Circuitry:

The BCH decoder is controlled by four signals: one to clear the decoder, one to clock data into the decoder with the feedback gate closed, one to clock data into the decoder with the feedback gate opened, and one to clock data out of the decoder. Each control signal must be isolated from all other control signals to the BCH decoder. Isolation of the control signals is accomplished through a 74139 Dual 2 to 4 Decoder. The decoder enable input is tied to DS* linking

the decoder function directly to the Z-80 DART and isolating the decoder from any S-100 internal control functions.

Pins A1 and B1 of the decoder are tied to address lines A14 and A15, neither of which are used by the Z-80 DART's address decoder. Unlike the encoder control, the decoder controller uses all state combinations of A14 and A15 to provide control over the decoder operations. Since the 74139 control lines (Y0 through Y3) are asserted (low) for all state combinations of A14 and A15, the BCH decoder is active when addresses 0000, 4000, 8000, and c000 (hex) are asserted. DART operations must be disabled (Tx disabled) because the DART will misinterpret OUT commands at the BCH decoder address as an instruction to load its transmit buffer. The DART will end up loading its buffer with erroneous data and transmitting it to the final bit sink.

The BCH decoder function is not affected whether the DART transmitter is disabled or not. Data is routed into the decoder through DO7 of the S-100 bus and is not affected by DART operations. Data is routed from the BCH decoder into the Z-100 through DI0. DI0 is multiplexed from both the BCH decoder and Z-80 DART to insure that input data is always taken from one source at a time.

The decoder clock and decoder clear functions are narrowed by combining pWR* on the S-100 bus with the appropriate 74139 decoder output lines. The narrowed control pulses are necessary to prevent inadvertent

triggering of the 74139 output lines when DS* transitions during changes A14 and A15. After control lines Y1 and Y2 (clock with gate closed and clock with gate open respectively) are narrowed, they are combined through a 7406 dual input And gate resulting in a single clock pulse transmitted to the BCH decoder at any given time.

Schematics:

Schematics of the BCH decoder and BCH decoder controller are found in Figures 13 and 14 respectively.

Test Mehodology:

Testing of the BCH decoder is conducted in two phases. The first phase determines whether the BCH decoder is decoding a code word to give the correct source word. The second phase determines whether the BCH decoder is generating the correct syndrome of a given generator polynomial and error polynomial. Each phase of testing is describe in greater detail in the paragraphs below.

Phase One: A software routine is written that prompts the user for the generator polynomial, n-k, and k. The generator polynomial is entered by the user as the bitwise representation of an ASCII character - 30h. For example, the bitwise representation of the generator polynomial x^3 + 1 is 1001. This amounts to the ASCII character 9 - 30h (in

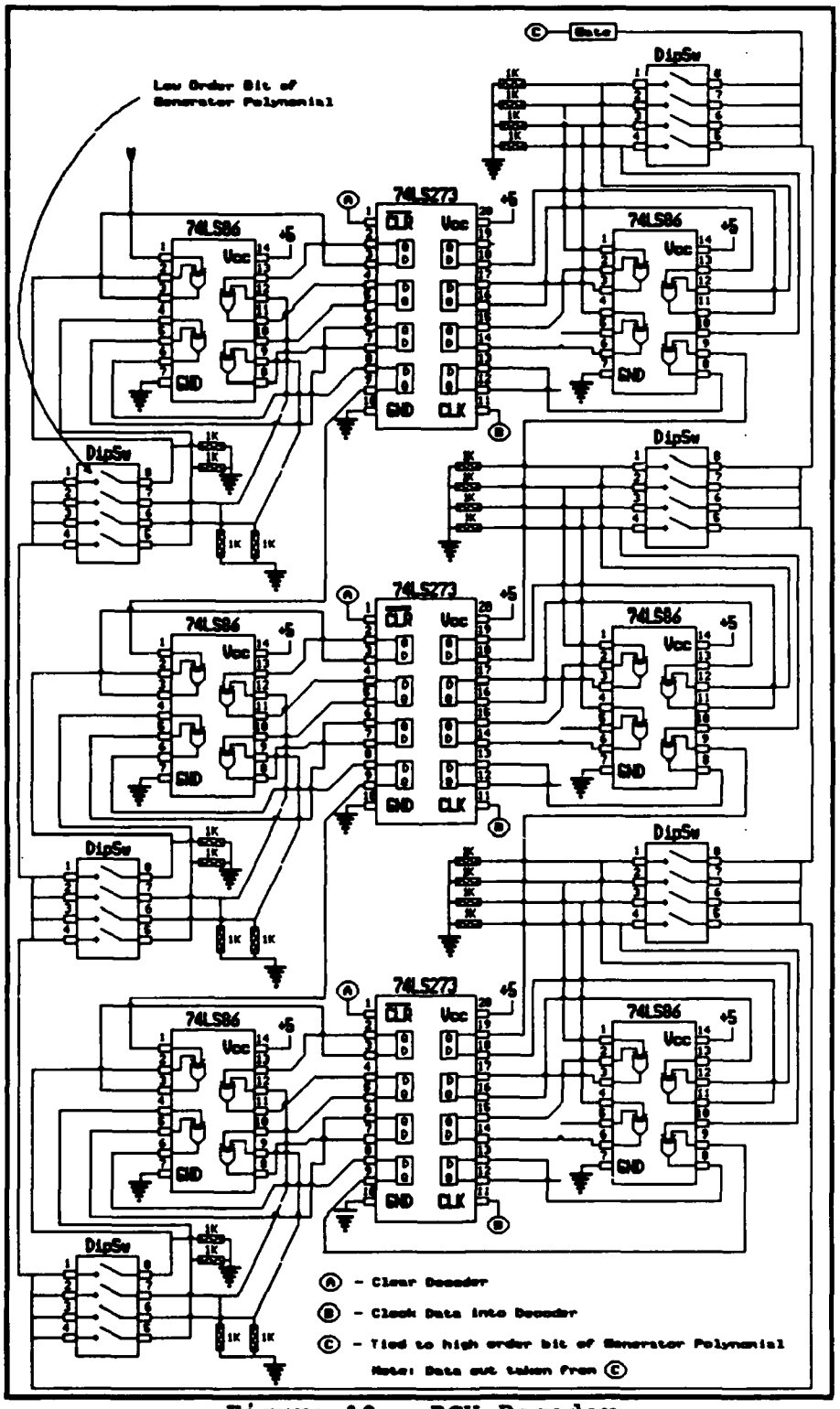

Figure 13 - BCH Decoder

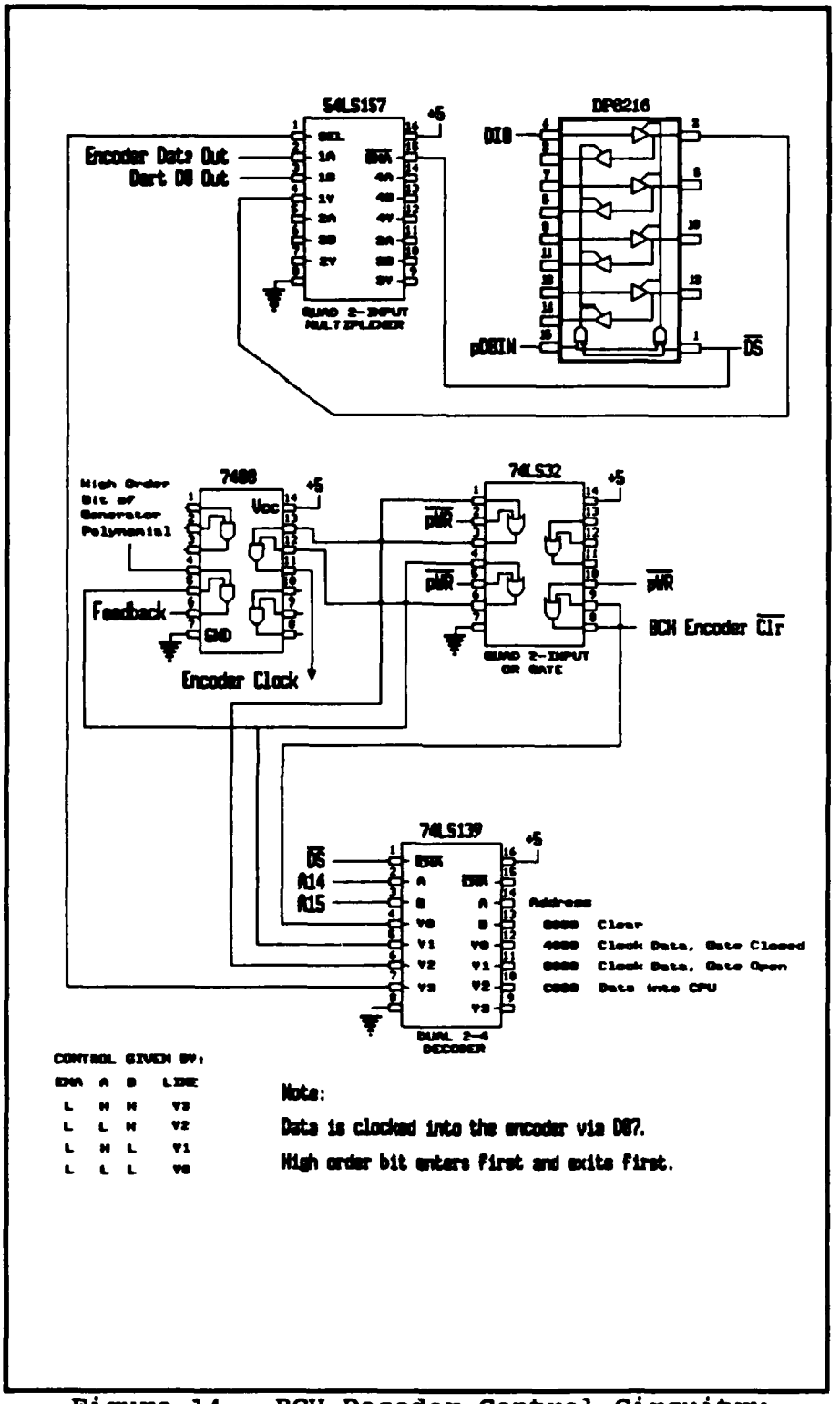

Figure 14 - BCH Decoder Control Circuitry

binary: 01101001 - 01100000). The generator polynomial chosen by the user must be enter as its bitwise value on the decoder dip switches. Values of n-k and k are entered by the user as their actual numeric value (the software will automatically convert to binary for internal usage).

After the generator polynomial, n-k, and k are entered, the software will prompt the user for a source word (any ASCII character will do as long as k = 7). The software encodes the source word internally using the generator polynomial to yield the code word. The code word is sent to the decoder for decoding, yielding the source word upon completion. The original source word, the binary representation of the code word, the binary representation of the decoded code word, and the final ASCII representation of the quotient (new source word) are displayed for comparison. Both the original and new source word must be identical to indicate the proper operation of the decoder. The software written for phase one can be found in appendix F.

Phase Two: A second software routine is written that uses a BCH (15,7) two error correcting code with $x^8 + x^7 + x^6 + x^4 + 1$ as the generator polynomial. The software generates all the error polynomials and concomitant syndromes for the given BCH code and displays them as a hard copy on the printer.

A third software routine is written that uses the same BCH (15,7) two error correcting code and $x^8 + x^7 + x^6 + x^4 + 1$ as the generator polynomial. Once again, the software generates all the error polynomials associated with the BCH code. Then the software sends each error polynomial to the decoder for decoding. Each resultant syndrome for the decoding process is stored with its concomitant error polynomial. The final list of error polynomials and syndromes are sent to a temporary file that can be printed as a hard copy. This final hard copy must match the original printed copy generated from the second software routine to indicate the proper operation of the decoder. The software written for phase two can be found in appendix G.

A final demonstration of both the decoder and encoder working as a complete product is described in chapter six. The final demonstration uses the syndrome created by the decoder to correct errors which ultimately yields the correctly decoded source word.

Test Results:

The decoder worked well for all generator polynomials chosen in Phase One. All syndromes for the (15,7) two error correcting BCH code generated by the decoder matched those generated in software during Phase Two.

Conclusions and Recommendations:

The decoder, as implemented, will work well for any cyclic code of order 23 or less. Because the Z-100 has 8 and 16 bit registers, it is most convenient to use codes of these lengths.

VI. Demonstration and Follow-on Thesis Effort

Demonstration:

Introduction: It is desirable, especially to those who might want to use the encoder and decoder in a follow-on effort, to prove that the final system design works properly. To this end, a "demonstration" software module was written which exercises the final system (comprised of the encoder hardware, decoder hardware, and DART interfaces) in a manner consistent with a realtime encoding and decoding problem. The problem, in essence, is to take an ASCII character, encode it on one computer, send the encoded word to a second computer, have the second computer decode the code word, display the result, and send a request for another encoded word back to the first computer, where upon the process starts over again. The demonstration is limited to the (15,7) two error correcting BCH code that has been discussed throughout this document.

System Configuration: The system is configured as shown in Figure 15. Only two computers are needed, one for the encoder and one for the decoder, because the communications channel is not simulated independently of the encoder function for the demonstration.

The encoder host computer's AUX output is tied to the DART input of the decoder board. The encoded word is sent over this link. The decoder host computer's AUX output is tied to the DART input of the encoder board. The request by the decoder for another encoded word is sent over this link. Although the system could be configured from DART to DART for both links, the current configuration proves that ANY computer with a serial output set at 9600 Baud, no parity, and one stop bit, can effectively communicate with the encoder and decoder boards. This means that the final system configuration can use any available PC computer as the host computer for the channel simulator.

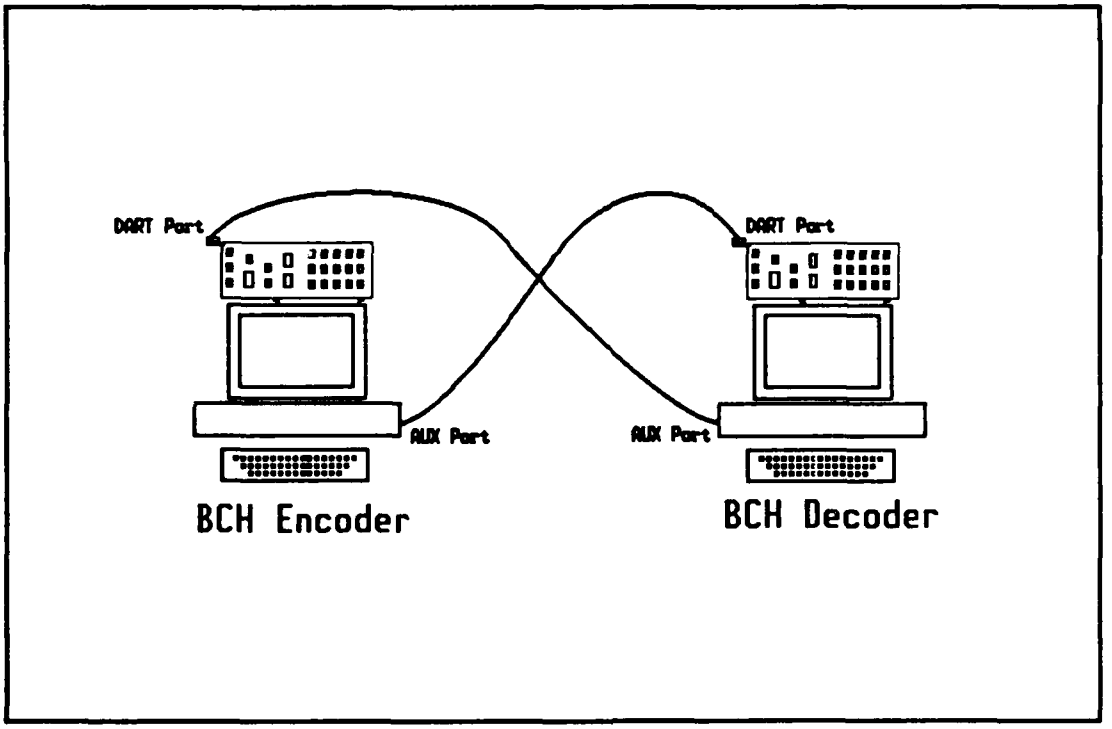

Figure 15 - System Configuration for Demonstration

Demonstration Software: The demonstration software written to exercise the encoder and decoder can best be understood in a step by step approach. The encoder and decoder are treated separately in the paragraphs below:

Encoder:

Step One: Initialize the DART to 9600 Baud, one stop bit, and no parity.

Step Two: Prompt the user for a source word. The user may enter any ASCII character on the keyboard.

Step Three: Encode the character and display the code word on the screen.

Step Four: Prompt the use to enquire of the users desire to add a error polynomial. If the answer is yes, accept an error polynomial from the screen. If the answer is not jump to step Six.

Step Five: Exclusive or the error polynomial to the encoded word.

Step Six: Transmit the final encoded word. Then wait for a request for another encoded word before repeating the process.

Decoder:

Step One: Initialize the DART to 9600 Baud, one stop bit and no parity.

Step Two: Generate the syndromes for the

(15,7) BCH two error correcting code and store the results in a look-up table.

Step Three: Receive an encoded word from the encoder.

Step Four: Decode the encoded word. If no syndrome results, display the transmitted word, and the final result (then skip to step seven).

Step Five: Display the resulting syndrome on the CRT, then use the syndrome and the look-up table to find the error polynomial. Exclusive-or the error polynomial to the transmitted word to form the corrected transmitted word.

Step Six: Decode the corrected transmitted word and determine if there is a syndrome. If there is a syndrome, display an error message and skip to step eight.

Step Seven: Display the final result on the CRT.

Step Eight: Transmit a request for another encoded word to the encoder. Go to step three.

Follow-On Thesis Effort:

As discussed in the overview of chapter one, several follow-on thesis efforts are required in order to meet the needs of the Foreign Technology Division. As part of the follow-on effort, and of immediate interest, is the simulation of a communications channel in software. The simulated channel should minimally address random gaussian

and Markov burst error noise patterns. As an aid in understanding the level of effort involved in developing the simulated communications channel, the following list of tasks to complete has been compiled.

A. Research the selected communication channels and associated noise (random gaussian and Markov burst error) to determine the probability of bit error (and bit error pattern) for each communications channel.

B. Select a computer and programming language suitable for writing the channel simulation programs needed. (I would recommend a Z-248 as the computer and C as the programming language).

C. Write the software routines that will ultimately simulate the communications channel.

D. Interface the selected computer and the software to simulate the communications channel with the existing encoder and decoder.

E. Write any driving routines that are required to provide a completely functional system.

F. Write your thesis, and demonstrate the final system to your thesis advisory board.

Appendiz A: 74747 D Flip Flop Interface Software

```
;*****************************************************************
;     STEP ONE - DRIVER ONE
;
;      Accepts a character from the keyboard and transmits it
;out the output port to the input port.  Receives the
;character from the input port and displays the character on
;the screen.  A visual check of the screen will show whether
;the transmission is handled correctly.
;
;*****************************************************************
;
;  Define constants
;
bell        equ         07h                    ;ASCII char for bell
cr          equ         0dh                    ;ASCII char for ;
;*****************************************************************
;
stack_area      segment     stack      ;define the stack
                                       ;area
                db          40 dup(?)   ;set aside room for stack
stack_area      ends
;
;
data_area       segment      ;define data area
outco           db          00h
inco            db          00h
char            db          00h
mych            db          41h
data_area       ends
;
;*****************************************************************
;
;
LAB_1           segment      ;define segment area for program
main            proc        far
                assume      cs:LAB_1, ds:data_area,
ss:stack_area
;
;set all register to zero (makes it easier to debug)
;
begin:          sub         ax,ax                  ;set ax to zero
                sub         bx,bx                  ;set bx to zero
                sub         cx,cx                  ;set cx to zero
                sub         dx,dx                  ;set dx to zero
;
;set DS reg to point to the current data area
;
    mov         ax,data_area          ;point to data_area
    mov         ds,ax                 ;DS now points to
                                           ;data_area
```

```
;
;begin part1 of lab - initialize data ready and data request
;lines
;
    mov       dl,01h              ;output port address
    in        al,dx               ;clear data ready
    mov       dl,03h              ;input port address
    in        al,dx               ;clear data request

;
;load the character to transmit into bx, and set data
request ;high
;
     mov       bl,mych             ;character in bx
     mov       dl,03h              ;input port address
     out       dx,al               ;set data request
                                   ;clear data ready
     mov       outco,08h           ;set out count number
     mov       inco,08h            ;set in count number

;
;begin output loop
;
Outlp:    mov       al,outco            ;load out count
                                        ;number
          cmp       al,00h              ;compare to zero
          jz        Inlp                ;jump to inloop
          mov       dl,00h              ;output port
          in        al,dx               ;check if data
                                        ;request
                                        ;is set
          and       al,08h              ;mask for third bit
          cmp       al,08h              ;is data request
                                        ;high?
          jne       outlp               ;wait until data
                                        ;request goes high
          mov       al,bl               ;load al with bit
          out       dx,al               ;data rdy high
                                        ;data req low
          shr       bl,1                ;shift bl right one
                                        ;bit
          mov       cl,outco            ;load out count
                                        ;number
          dec       cx                  ;dec by one
          mov       outco,cl    ;replace count number
;
;Begin inloop:
;
Inlp:   mov       dl,02h              ;input port
        in        al,dx               ;get some data
        and       al,05h              ;mask for rdy and
                                      ;data
```

```
        cmp     al,02h          ;less than 2?
        jl      Inlp            ;wait for data ready
        and     al,01h          ;get data bit
        mov     cl,char         ;put character into
                                ;cx
        mov     ch,al           ;put bit into cl
        shr     cx,1            ;shift bit into char
        mov     char,cl         ;put character back
        or      cl,al           ;append data bit
        mov     al,inco         ;enough bits yet?
        cmp     al,01h          ;last bit?
        je      Prnt            ;yes, go print
        dec     al              ;decrease incount by
                                ;one
        mov     inco,al         ;store the count
        mov     dl,03h          ;input port
        out     dx,al           ;set data request,
                                ;clear
                                ;data ready
        jmp     Outlp           ;jump to outloop
;
;Time to print the received character
;
Prnt:   mov     dl,char         ;dl has character
        mov     ah,02h
        int     21h

;All do, so exit from program
; Exit3:        mov     ax,4c00h          ;use DOS
function 4c
;with return
              int     21h               ;to exit from
program
;
main          endp
LAB_1         ends
              end     main              ;end assy
```

```
;*****************************************************************
;     STEP ONE - DRIVER TWO
;
;     Take a single character and transmit it out the output
;port to the input port.  Receive the character from the
;input port and check the character internally to determine
;if the received character matches the sent character.
;Report the number of characters sent and the number of
;errors detected to the CRT.  Repeat the process
;indefinitely.  (Note that an overflow condition will arise
;when the number of bits sent equals about 25k bits.
;Running the program four times consecutively amounts to
;transmitting over 10M bits.)
;
;*****************************************************************
;
;  Define constants
;
bell            equ       07h           ;ASCII char for bell
cr              equ       0dh           ;ASCII char for carriage
                                        ;return
lf              equ       0ah           ;ASCII char for line
feed
;
;
;*****************************************************************
;
stack_area      segment   stack         ;define the stack area
                db        40 dup(?)     ;set aside room for
stack
stack_area      ends
;
;
data_area       segment                 ;define data area
msg1            db        'and','$'
msg2            db        'cr,'$'
outco           db        00h
inco            db        00h
char            db        00h
mych            db        41h
good            dw        0000h
bad             dw        0000h
data_area       ends
;
;*****************************************************************
;
;
LAB_1           segment                 ;define segment area for
program
main            proc      far
                assume    cs:LAB_1, ds:data_area,
ss:stack_area
```

```
;
;set all register to zero (makes it easier to debug)
;
begin:          sub         ax,ax          ;set ax to zero
                sub         bx,bx          ;set bx to zero
                sub         cx,cx          ;set cx to zero
                sub         dx,dx          ;set dx to zero
;
;set DS reg to point to the current data area
;
                mov         ax,data_area     ;point to data_area
                mov         ds,ax            ;DS now points to
data_area
;
;begin part1 of lab - initialize data ready and data request
;lines
;
                mov         dl,01h         ;output port address
                in          al,dx          ;clear data ready
                mov         dl,03h         ;input port address
                in          al,dx          ;clear data request

;
;load the character to transmit into bx, and set data
request ;high
;
loop1:          mov         bl,mych        ;character in bx
                mov         dl,03h         ;input port address
                out         dx,al          ;set data request
                                           ;clear data ready
                mov         outco,08h      ;set out count number
                mov         inco,08h       ;set in count number

;
;begin output loop
;
Outlp:          mov         al,outco       ;load out count number
                cmp         al,00h         ;compare to zero
                jz          Inlp           ;jump to inloop
                mov         dl,00h         ;output port
                in          al,dx          ;check if data request
                                           ;is set
                and         al,08h         ;mask for third bit
                cmp         al,08h         ;is data request high?
                jne         outlp          ;wait until data request

                                           ;goes high
                mov         al,bl          ;load al with bit
                out         dx,al          ;transmit bit, data
                                    ;ready
                                           ;goes high, data req.
                                      ;low
```

```
                shr         bl,1          ;shift bl right one bit
                mov         cl,outco      ;load out count number
                dec         cx            ;dec by one
                mov         outco,cl      ;replace count number
;
;Begin inloop:
;
Inlp:           mov         dl,02h        ;input port
                in          al,dx         ;get some data
                and         al,05h        ;mask for rdy and data
                cmp         al,02h        ;less than 2?
                jl          Inlp          ;wait for data ready
                and         al,01h        ;get data bit
                mov         cl,char       ;put character into cx
                mov         ch,al         ;put bit into cl
                shr         cx,1          ;shift bit into char
                mov         char,cl       ;put character back
                or          cl,al         ;append data bit
                mov         al,inco       ;enough bits yet?
                cmp         al,01h        ;last bit?
                je          Prnt          ;yes, go print
                dec         al            ;decrease incount by one

                mov         inco,al       ;store the count
                mov         dl,03h        ;input port
                out         dx,al         ;set data request, clear

                                          ;data ready
                jmp         Outlp         ;jump to outloop
;
;Time to print the results.
;
Prnt:           mov         dl,char       ;dl has character
                mov         al, mych      ;al has sent character
                cmp         al,dl         ;compare characters
                je          upgd          ;same? inc good count
                mov         cx,bad        ;different?
                inc         cx            ;inc bad count
                mov         bad,cx        ;store bad count
                jmp         pt            ;print results
upgd:           mov         cx,good       ;get the current good
                                          ;count
                inc         cx            ;increment the good
                                          ;count
                mov         good,cx       ;store the good count
pt:             mov         bx,good       ;place the good count in
                                          ;bx
                call        binidec       ;call to convert to
                                          ;decimal
                mov         dx,offset msg1      ;insert and in
                                          ;output
                mov         ah,09h        ;DOS call to print
```

```
                                ;string
                int             21h
                mov             bx,bad          ;place bad count into bx

                call            binidec         ;convert bx to binary
                mov             dx,offset msg2          ;output a
                                ;carriage return
                mov             ah,09h
                int             21h
                jmp             loop1           ;repeat the process

;*************************************************************
;Binidec program                 number in bx
;*************************************************************

binidec         proc            near
                push            bx
                push            cx
                mov             cx,10000d
                call            dec_div
                mov             cx,1000d
                call            dec_div
                mov             cx,100d
                call            dec_div
                mov             cx,10d
                call            dec_div
                mov             cx,1d
                call            dec_div
                pop             cx
                pop             dx
                ret

dec_div         proc            near
                mov             ax,bx
                cwd
                div             cx
                mov             bx,dx
                mov             dl,al
                add             dl,30h
                mov             ah,02h          ;display character
                int             21h
                ret

dec_div         endp
binidec         endp


;All do, so exit from program
;
exit3:          mov             ax,4c00h        ;use DOS function 4c
                int             21h             ;to exit from program
;
```

```
main            endp
LAB_1           ends
                end         main          ;end assy
```

```
;*****************************************************************
;     STEP TWO - FIRST Z-100
;
;     Accept a character from the keyboard, echo it to the
;screen, and transmit the character out the output port to
;the second Z-100. Watch the input port to see if the data
;ready line goes high.  When the data ready line goes high,
;receive the character from the input port and echo the
;character to the screen.  It should be the same character
;as the one initially sent.  Repeat this process
;indefinitely.
;
;*****************************************************************
;
;  Define constants
;
bell            equ       07h           ;ASCII char for bell
cr              equ       0dh           ;ASCII char for carriage
return
lf              equ       0ah           ;ASCII char for line
feed
;
;
;*****************************************************************
;
stack_area      segment   stack         ;define the stack area
                db        40 dup(?)     ;set aside room for
stack
stack_area      ends
;
;
data_area       segment                 ;define data area
outco           db        00h
inco            db        00h
char            db        00h
mych            db        41h
data_area       ends
;
;*****************************************************************
;
;
LAB_1           segment                 ;define segment area
main            proc      far
                assume    cs:LAB_1, ds:data_area,
ss:stack_area
;
;set all register to zero (makes it easier to debug)
;
begin:          sub       ax,ax         ;set ax to zero
                sub       bx,bx         ;set bx to zero
                sub       cx,cx         ;set cx to zero
                sub       dx,dx         ;set dx to zero
```

```
;
;set DS reg to point to the current data area
;
                mov     ax,data_area    ;point to data_area
                mov     ds,ax           ;DS now points to
data_area
;
;begin part1 of lab - initialize data ready and data request
lines
;
                mov     dl,01h          ;output port address
                in      al,dx           ;clear data ready
                mov     dl,03h          ;input port address
                in      al,dx           ;clear data request

;
;load the character to transmit into bx, and set data
request high
;
loop1:          mov     ah,01h          ;read char and echo
                int     21h             ;DOS call

                mov     bl,al           ;character in bx
                mov     outco,08h       ;set out count number

;
;begin output loop
;
Outlp:          mov     dl,00h          ;output port
                in      al,dx           ;check if data request
                                        ;is set
                and     al,08h          ;mask for third bit
                cmp     al,08h          ;is data request high?
                jne     outlp           ;wait until data request
                                        ;goes high
                mov     al,bl           ;load al with bit
                out     dx,al           ;transmit bit, data RDY
                                        ;goes high, data req.
                                    ;low
                shr     bl,1            ;shift bl right one bit
                mov     cl,outco        ;load out count number
                dec     cx              ;dec by one
                mov     outco,cl        ;replace count number
                cmp     cl,00h          ;all done?
                jne     outlp           ;no, transmit next bit
                mov     inco,08h        ;initialize incount to 8
                jmp     inlp            ;go get a character
;
;Begin inloop:
;
Inlp:           mov     dl,03h          ;set data request
                out     dx,al           ;and clear data ready
```

```
Wt:             mov     dl,02h      ;input port
                in      al,dx       ;get some data
                and     al,05h      ;mask for rdy and data
                cmp     al,02h      ;less than 2?
                jl      Wt          ;wait for data ready
                and     al,01h      ;get data bit
                mov     cl,char     ;put character into cx
                mov     ch,al       ;put bit into cl
                shr     cx,1        ;shift bit into char
                mov     char,cl     ;put character back
                mov     al,inco     ;enough bits yet?
                cmp     al,01h      ;last bit?
                je      Prnt        ;yes, go print
                dec     al          ;decrease incount by one
                mov     inco,al     ;store the count
                jmp     Inlp        ;jump to inloop
;
;Time to print the received character
;
Prnt:           mov     dl,char     ;dl has character
                mov     ah,02h
                int     21h
                mov     outco,08h   ;initialize outcount
                jmp     loop1       ;get another character

;All do, so exit from program
;
exit3:          mov     ax,4c00h    ;use DOS function 4c
                int     21h         ;to exit from program
;
main            endp
LAB_1           ends
                end     main        ;end assy
```

```
;*****************************************************************
;     STEP 2 - SECOND Z-100
;
;     Receive a character from the input port, echo it to
;the screen and output this character to the output port.
;Repeat the process indefinitely.
;
;*****************************************************************
;  Define constants
;
bell            equ       07h           ;ASCII char for bell
cr              equ       0dh           ;ASCII char for carriage
                                        ;return
lf              equ       0ah           ;ASCII char for line
feed
;
;
;*****************************************************************
;
stack_area      segment   stack         ;define the stack area
                db        40 dup(?)     ;set aside room for
stack
stack_area      ends
;
;
data_area       segment                 ;define data area
outco           db        00h
inco            db        00h
char            db        00h
mych            db        41h
data_area       ends
;
;*****************************************************************
;
;
LAB_1           segment                 ;define segment area
main            proc      far
                assume    cs:LAB_1, ds:data_area,
ss:stack_area
;
;set all register to zero (makes it easier to debug)
;
begin:          sub       ax,ax         ;set ax to zero
                sub       bx,bx         ;set bx to zero
                sub       cx,cx         ;set cx to zero
                sub       dx,dx         ;set dx to zero
;
;set DS reg to point to the current data area
;
                mov       ax,data_area  ;point to data_area
                mov       ds,ax         ;DS now points to
data_area
```

```
;
;begin part1 of lab - initialize data ready and data request
;lines
;
                mov     dl,01h      ;output port address
                in      al,dx       ;clear data ready
                mov     dl,03h      ;input port address
                in      al,dx       ;clear data request

;
;load the character to transmit into bx, and set data
request ;high
;
                mov     inco,08h    ;set out count number
                jmp     inlp        ;go get a character
;
;begin output loop
;
Outlp:          mov     dl,00h      ;output port
                in      al,dx       ;check if data request
                                    ;is set
                and     al,08h      ;mask for third bit
                cmp     al,08h      ;is data request high?
                jne     outlp       ;wait until data request

                                    ;goes high
                mov     al,bl       ;load al with bit
                out     dx,al       ;transmit bit, data
                                ;ready high, data req
                                    ;low
                shr     bl,1        ;shift bl right one bit
                mov     cl,outco    ;load out count number
                dec     cx          ;dec by one
                mov     outco,cl    ;replace count number
                cmp     cl,00h      ;all done?
                jne     outlp       ;no, transmit next bit
                mov     inco,08h    ;initialize incount to 8

                jmp     inlp        ;go get a character
;
;Begin inloop:
;
Inlp:           mov     dl,03h      ;set data request
                out     dx,al       ;and clear data ready

Wt:             mov     dl,02h      ;input port
                in      al,dx       ;get some data
                and     al,05h      ;mask for rdy and data
                cmp     al,02h      ;less than 2?
                jl      Wt          ;wait for data ready
                and     al,01h      ;get data bit
                mov     cl,char     ;put character into cx
```

```
                mov     ch,al           ;put bit into cl
                shr     cx,1            ;shift bit into char
                mov     char,cl         ;put character back
                mov     al,inco         ;enough bits yet?
                cmp     al,01h          ;last bit?
                je      Prnt            ;yes, go print
                dec     al              ;decrease incount by one

                mov     inco,al         ;store the count
                jmp     Inlp            ;jump to inlp
;
;Time to print the received character
;
Prnt:           mov     dl,char         ;dl has character
                mov     ah,02h
                int     21h
                mov     outco,08h       ;initialize outcount
                mov     bl,char
                jmp     outlp

;All do, so exit from program
;
exit3:          mov     ax,4c00h        ;use DOS function 4c
                int     21h             ;to exit from program
;
main            endp
LAB_1           ends
                end     main            ;end assy
```

Appendix B - Protocol Software

```
;*****************************************************************
;PROTOCOL 2
;
;     Protocol 2 is performs the same function as STEP 2-1 -
;FIRST Z-100. That is, it transmits a character to another
;Z-100 and receives that character back.  Both the
;transmitted character and the received character are
;displayed on the screen so that they can be compared.
;Protocol 2 adds protocol in the form of software checks to
;insure that the transmission occurs without error.  The
;original transmitted character is outputted twice to insure
;that the data ready line is clocked high.  After eight bits
;have been transmitted (one character), Protocol two will
;check to see if all eight bits have been received by the
;second Z-100.  Protocol 2 does this by waiting a specified
;time,then sampling the data request line.  If the data
;request line is high all eight bits of the word were not
;received.  Protocol two then prints the message "Trapout"
;indicating that it is about to repeat the original
;transmission.  Protocol two will again wait a specified
;time giving the second Z-100 sufficient time to reset
;itself for a new character, then the character is
;retransmitted.
;     When Protocol two is in the receive mode, it checks
;the data ready line twice to insure that the data is stable
;on the data line.  Protocol two will count the number of
;bits it receives, and if sufficient bits are not received
;in the given amount of time it prints the message "Trapin"
;indicating that it is going to start the process of
;receiving a character over ;again.
;
;*****************************************************************
;
;  Define constants
;
bell            equ       07h           ;ASCII char for bell
cr              equ       0dh           ;ASCII char for CR
lf              equ       0ah           ;ASCII char for line
feed
;
;
;*****************************************************************
;
stack_area      segment   stack         ;define the stack area
                db        40 dup(?)     ;set aside room for
                                        ;stack
stack_area      ends
;
;
data_area       segment                 ;define data area
```

```
msg1            db          'Trapout','$'
msg2            db          'Trapin','$'
msg3            db          bell,cr,lf,lf
                db          'I/O at port 02h!',cr,lf,lf,'$'
outco           db          00h
inco            db          00h
char            db          00h
mych            db          41h
data_area       ends
;
;*****************************************************************
;
;
LAB_1           segment                 ;define segment area
main            proc        far
                assume      cs:LAB_1, ds:data_area,
            ss:stack_area
;
;set all register to zero (makes it easier to debug)
;
begin:          sub         ax,ax       ;set ax to zero
                sub         bx,bx       ;set bx to zero
                sub         cx,cx       ;set cx to zero
                sub         dx,dx       ;set dx to zero
;
;set DS reg to point to the current data area
;
                mov         ax,data_area    ;point to data_area
                mov         ds,ax       ;DS now points to
data_area
;
;begin part1 of lab - initialize data ready and data request
;lines
;
                mov         dl,01h      ;output port address
                in          al,dx       ;clear data ready
                mov         dl,03h      ;input port address
                in          al,dx       ;clear data request

;
;load the character to transmit into bx, and set data
request ;high
;
loop1:          mov         al,51h      ;load al with the char
                mov         bl,al       ;character in bx
                mov         outco,08h   ;set out count number

;
;begin output loop
;
Outlp:          mov         dl,00h      ;output port
                in          al,dx       ;check if data request
```

```
                                        ;is high
                and         al,08h      ;mask for third bit
                cmp         al,08h      ;is data request high?
                jne         outlp       ;wait until data request
                                        ;goes high
O2:             in          al,dx       ;check data request
                and         al,08h      ;mask for third bit
                cmp         al,08h      ;is data request high?
                jne         O2          ;if not, wait until it
                                        ;does go high
                mov         al,bl       ;put bit into al
                out         dx,al       ;transmit bit, data
                                        ;ready high, data req
                                        ;low
                shr         bl,1        ;shift bl right one bit

                mov         cl,outco    ;load out count number
                dec         cx          ;dec by one
                mov         outco,cl    ;replace count number
                cmp         cl,00h      ;all done?
                jne         outlp       ;no, transmit next bit
                mov         inco,08h    ;initialize incount to 8


;was the data received?  Is data request still high?
;
                mov         cx,0020h    ;wait about twenty
Wait2:          loop        Wait2       ;operations

                mov         dl,00h      ;output port
                in          al,dx       ;get data request bit
                and         al,08h      ;mask for data request
                cmp         al,08h      ;is it high
                jne         inlp2       ;no? we're OK, go on to
                                        ;receive the char
                mov  dx,offset msg1     ;data request is still
                                      ;high
                mov         ah,09h      ;so output "trapout"
                int         21h
                mov         cx,1000h    ;wait for the other
                                    ;Z-100
Wait1:          loop        Wait1       ;to reset itself
                jmp         loop1       ;go retransmit the char
;
;Begin inloop:
;

Inlp:           mov         dl,03h      ;set data request
                out         dx,al       ;and clear data ready
                out         dx,al       ;again, just to be sure
P:              mov         cx,0000h    ;clear cx (the timer)
```

```
Wt:             mov     dl,02h      ;input port
                in      al,dx       ;get some data
                and     al,05h      ;mask for rdy and data

                inc     cx          ;increment cx
                cmp     cx,100h     ;is time up?
                je      Prob        ;yes, jump to Prob(lem)

                cmp     al,02h      ;less than 2?
                jl      Wt          ;wait for data ready

Wt2:            mov     dl,02h      ;check for data ready
                                    ;again
                in      al,dx       ;get some data
                and     al,05h      ;mask for rdy and data

                inc     cx          ;increment the timer
                cmp     cx,100h     ;time up yet?
                je      Prob        ;yes, jump to Prob(lem)

                cmp     al,02h      ;less than 2?
                jl      Wt2         ;wait for data rdy

                and     al,01h      ;get data bit
                mov     cl,char     ;put character into cx
                mov     ch,al       ;put bit into cl
                shr     cx,1        ;shift bit into char
                mov     char,cl     ;put character back
                mov     al,inco     ;enough bits yet?
                cmp     al,01h      ;last bit?
                je      Prnt        ;yes, go print
                dec     al          ;decrease incount by one

                mov     inco,al     ;store the count
                jmp     Inlp        ;jump to inloop

Prob:           mov     inco,08h    ;reinitialize inco
                mov     dx,offset msg2   ;print "trapin"
                mov     ah,09h      ;using DOS call 09h
                int     21h
                mov     cx,0200h    ;now wait a little bit
Wait4:          loop    Wait4       ;to make sure the other

                jmp     P           ;Z-100 has reinitialized


;
;Time to print the received character
;
Prnt:           mov     dl,char     ;dl has character
                mov     ah,02h
                int     21h
```

```
                mov         outco,08h    ;initialize outcount
                jmp         loop1

;All do, so exit from program
;
exit3:          mov         ax,4c00h     ;use DOS function 4c
                int         21h          ;to exit from program
;
main            endp
LAB_1           ends
                end         main         ;end assy
```

```
;*****************************************************************
;     PROTOCOL 3.ASM
;
;     Protocol 3 is virtually identical to protocol two but
;is meant to run on the "other Z-100".  Protocol 3 performs
;the same function as STEP2-2.ASM.  That is, it receives a
;transmitter character, echoes it to the screen, and
;transmits the character back to the first Z-100.  Protocol
;3 starts its program in the receive mode.  Therefore, there
;is an initial jump to the Input loop (Inlp) early in the
;program sequence.
;
;*****************************************************************
;
;  Define constants
;
bell            equ       07h           ;ASCII char for bell
cr              equ       0dh           ;ASCII char for carriage
                                        ;return
lf              equ       0ah           ;ASCII char for line
feed
;
;
;*****************************************************************
;
stack_area      segment   stack         ;define the stack area
                db        40 dup(?)     ;set aside room for
stack
stack_area      ends
;
data_area       segment                 ;define data area
msg1            db        'Trapin!',cr,lf,'$'
msg2            db        'Trapout',cr,lf,'$'
msg3            db        bell,cr,lf,lf
                db        'I/O at port 02h!',cr,lf,lf,'$'
outco           db        00h
inco            db        00h
char            db        00h
mych            db        41h
data_area       ends
;
;*****************************************************************
;
;
LAB_1           segment                 ;define segment area
main            proc      far
                assume    cs:LAB_1, ds:data_area,
 ss:stack_area
;
;set all register to zero (makes it easier to debug)
;
begin:          sub       ax,ax         ;set ax to zero
```

```
                sub     bx,bx           ;set bx to zero
                sub     cx,cx           ;set cx to zero
                sub     dx,dx           ;set dx to zero
;
;set DS reg to point to the current data area
;
                mov     ax,data_area    ;point to data_area
                mov     ds,ax           ;DS now points to
                                        ;data_area
;
;begin part1 of lab - initialize data ready and data request
;lines
;
                mov     dl,01h          ;output port address
                in      al,dx           ;clear data ready
                mov     dl,03h          ;input port address
                in      al,dx           ;clear data request

;
;load the character to transmit into bx, and set data
;request high
;
                mov     inco,08h        ;set out count number
                jmp     inlp            ;go get a character
;
;begin output loop
;
Outlp:          mov     dl,00h          ;output port
                in      al,dx           ;check if data request
                                        ;is high
                and     al,08h          ;mask for third bit
                cmp     al,08h          ;is data request high?
                jne     outlp           ;wait until data request

                                        ;goes high
O2:             in      al,dx           ;check data request
                                    ;again
                and     al,08h          ;mask for third bit
                cmp     al,08h          ;is data request high?
                jne     O2              ;if not, wait until it
                                        ;does go high
                mov     al,bl           ;put bit into al
                out     dx,al           ;transmit bit, data
                                   ;ready high,data req
                                        ;low
                shr     bl,1            ;shift bl right one bit
                mov     cl,outco        ;load out count number
                dec     cx              ;dec by one
                mov     outco,cl        ;replace count number
                cmp     cl,00h          ;all done?
                jne     outlp           ;no, transmit next bit
                mov     inco,08h        ;initialize incount to 8
```

```
;was the data received?  Is data request still high?
;
                mov         cx,0020h     ;wait about twenty
Wait2:          loop        Wait2        ;operations

                mov         dl,00h       ;output port
                in          al,dx        ;get data request bit
                and         al,08h       ;mask for data request
                cmp         al,08h       ;is it high
                jne         inlp2        ;no? we're OK, go on to
                                         ;receive the char
                mov  dx,offset msg1      ;data request is still
                                       ;high
                mov         ah,09h       ;so output "trapout"
                int         21h

                mov         cx,1000h     ;wait for the other
                                       ;Z-100
Wait1:          loop        Wait1        ;to reset itself
                jmp         loop1        ;go retransmit the char
;
;Begin inloop:
;

Inlp:           mov         dl,03h       ;set data request
                out         dx,al        ;and clear data ready
                out         dx,al        ;again, just to be sure
P:              mov         cx,0000h     ;clear cx (the timer)

Wt:             mov         dl,02h       ;input port
                in          al,dx        ;get some data
                and         al,05h       ;mask for rdy and data

                inc         cx           ;increment cx
                cmp         cx,100h      ;is time up?
                je          Prob         ;yes, jump to Prob(lem)

                cmp         al,02h       ;less than 2?
                jl          Wt           ;wait for data ready

Wt2:            mov         dl,02h       ;check for data ready
                                       ;again
                in          al,dx        ;get some data
                and         al,05h       ;mask for rdy and data

                inc         cx           ;increment the timer
                cmp         cx,100h      ;time up yet?
                je          Prob         ;yes, jump to Prob(lem)

                cmp         al,02h       ;less than 2?
```

```
                jl          Wt2             ;wait for data rdy

                and         al,01h          ;get data bit
                mov         cl,char         ;put character into cx
                mov         ch,al           ;put bit into cl
                shr         cx,1            ;shift bit into char
                mov         char,cl         ;put character back
                mov         al,inco         ;enough bits yet?
                cmp         al,01h          ;last bit?
                je          Prnt            ;yes, go print
                dec         al              ;decrease incount by one

                mov         inco,al         ;store the count
                jmp         Inlp            ;jump to inloop

Prob:           mov         inco,08h        ;reinitialize inco
                mov         dx,offset msg2   ;print "trapin"
                mov         ah,09h          ;using DOS call 09h
                int         21h
                mov         cx,0200h        ;now wait a little bit
Wait4:          loop        Wait4           ;to make sure the other

                jmp         P               ;Z-100 has reinitialized


;
;Time to print the received character
;
Prnt:           mov         dl,char         ;dl has character
                mov         ah,02h
                int         21h
                mov         outco,08h       ;initialize outcount
                jmp         loop1

;All do, so exit from program
;
exit3:          mov         ax,4c00h        ;use DOS function 4c
                int         21h             ;to exit from program
;
main            endp
LAB_1           ends
                end         main            ;end assy
```

```
;*****************************************************************
;
;     DART_DRIVER_ONE
;
;     DART-DRIVER_ONE is used in conjunction with DART-
;DRIVER_TWO to test the performance of the DART (serial
;I/O port) on the BCH encoder.  DART_DRIVER_ONE sends a
;character to the DART through the modem (J2) serial I/O
;port.  J2 is configured to 9600 baud, one stop bit, no
;parity, and x16 clock.  After the character has been sent,
;DART_DRIVER_ONE receives the character back through the
;same port (J2).  The character received is compared to the
;character sent.  After a block of 1000 characters are sent,
;the results are tabulated and the number of errors found are
;displayed on the CRT.
;
;*****************************************************************
;
;  Define constants
;
bell            equ         07h         ;ASCII char for bell
cr              equ         0dh         ;ASCII char for carr
                                    ;return
lf              equ         0ah         ;ASCII char for line feed

port1           equ         00h         ;output port, data
port2           equ         01h         ;output port, control
char            equ         41h         ;ASCII char for "A"
;
;

;*****************************************************************
;
stack_area      segment     stack       ;define the stack area
                db          40 dup(?)   ;set aside room for
stack
stack_area      ends
;
;
data_area       segment                 ;define data area
count           dw          0000h       ;to count words tx
bad             dw          0000h       ;to count errors
msg1            db          'and','$'
msg2            db          cr,'$'
data_area       ends
;
;*****************************************************************
;
```

```
;
LAB_1           segment                 ;define segment area
main            proc        far
                assume      cs:LAB_1, ds:data_area,
ss:stack_area
;
;set all register to zero (makes it easier to debug)
;
begin:          sub         ax,ax               ;set ax to zero
                sub         bx,bx               ;set bx to zero
                sub         cx,cx               ;set cx to zero
                sub         dx,dx               ;set dx to zero
;
;set DS reg to point to the current data area
;
                mov         ax,data_area    ;point to data_area
                mov         ds,ax           ;DS points to
data_area
;
;Start the program here:
;
Loop1:      mov         bx,0000d        ;bx will count the char
                                    ;tx
;
;  Output the received data from the Z-100 using DOS call
;  04h
;

Loop2:      mov         dl,char         ;character loaded into dl
            mov         ah,04h          ;DOS call 04h outputs
            int         21h             ;the char through AUX
                                ;(J2)

;
;  Receive the charcter back through the same port (J2),
;using DOS call 03h.
;
            mov         ah,03h          ;DOS call to receive
            int         21h             ;a charcter through J2
                                        ;character is in al

;
;   Compare the received character to the transmitted
;charcter and tabulate the results.
;
            cmp         al,char         ;is character the same?
            jne         Ibad            ;no increment error count

Iword:      inc         bx              ;inc number of char sent
            cmp         bx,1000d        ;1000 words sent?
            je          Prnt            ;yes, output results
            jmp         Loop2           ;tx next character
```

```
;
Ibad:       mov         cx,bad          ;get current error count
            inc         cx              ;increment by one
            mov         bad,cx          ;replace error count
            jmp         Iword           ;go back to main routine
;
;
;Time to print the results.
;
Prnt:       mov         bx,count        ;bx contains the number
            inc         bx              ;of words x 1000
            mov         count,bx        ;store count
            call        binidec         ;convert to decimal and
                                        ;print the results

            mov    dx,offset msg1       ;insert and in output
            mov         ah,09h          ;DOS call to print string

            int         21h

            mov         bx,bad          ;place bad count into bx
            call        binidec         ;convert bx to binary

            mov    dx,offset msg2       ;output a carriage return

            mov         ah,09h
            int         21h
;
;    See if over 10M bits have been transmitted.  Stop the
;program when this happens.
;
            mov         cx,count        ;number of words tx in cx

            cmp         cx,1300d        ;1300 x 1000 x 8bits =
                                        ;10.4 M bits
            je          exit1           ;if equal, all done
            jmp         loop1           ;repeat the process

;*************************************************************
;Binidec program                  number in bx
;*************************************************************

binidec       proc        near
              push        bx
              push        cx
              mov         cx,10000d
              call        dec_div
              mov         cx,1000d
              call        dec_div
              mov         cx,100d
              call        dec_div
              mov         cx,10d
```

```
                call    dec_div
                mov     cx,1d
                call    dec_div
                pop     cx
                pop     dx
                ret

dec_div         proc    near
                mov     ax,bx
                cwd
                div     cx
                mov     bx,dx
                mov     dl,al
                add     dl,30h
                mov     ah,02h          ;display character
                int     21h
                ret

dec_div         endp
binidec         endp


;All do, so exit from program
;
exit1:          mov     ax,4c00h        ;use DOS function 4c
                int     21h             ;to exit from program
;
main            endp
LAB_1           ends
                end     main            ;end assy


;****************************************************************
;
;     DART_DRIVER_TWO
;
;     DART_DRIVER_TWO provides the test routines to
;determine if the DART is receiving a character correctly
;from another Z-100.  Once the character is received, the
;DART outputs the character back to the Z-100.
;
;****************************************************************
;
;  Define constants
;
bell            equ     07h     ;ASCII char for bell
cr              equ     0dh     ;ASCII char for carriage
return
lf              equ     0ah     ;ASCII char for line feed
port1           equ     00h     ;output port, data
port2           equ     01h     ;output port, control
;
```

```
;
;*****************************************************************
;
stack_area      segment   stack     ;define the stack area
                db        40 dup(?)    ;set aside room for
stack
stack_area      ends
;
;
data_area       segment                ;define data area

data_area       ends
;
;*****************************************************************
;
;
LAB_1           segment             ;define segment area for
program main            proc       far
                assume    cs:LAB_1, ds:data_area,
           ss:stack_area
;
;set all register to zero (makes it easier to debug)
;
begin:          sub       ax,ax                ;set ax to zero
                sub       bx,bx                ;set bx to zero
                sub       cx,cx                ;set cx to zero
                sub       dx,dx                ;set dx to zero
;
;set DS reg to point to the current data area
;
     mov        ax,data_area         ;point to data_area
     mov        ds,ax                ;DS now points to
                              ;data_area
;
;
;  Initialize the DART to x16 clock, 1 stop bit, 8 Tx/Rx
;bits, no parity, and no interrupts (polled mode).
;
;       AL contains the control byte to the DART
;       DX contains the control byte address

                push      ax             ;save ax
                push      dx             ;save dx
                mov       dx,port2       ;output port control
                                         ;address
                mov       al,18h         ;channel reset command

                out       dx,al          ;issue command
                out       dx,al          ;again, just to be
                                       ;sure
WR1:            mov       al,01h         ;point to WR1
                out       dx,al          ;load pointer into WR0
```

```
                mov     al,00h          ;disable all
                                        ;interrupts
                out     dx,al           ;issue command
WR2:            mov     al,02h          ;point to WR2
                out     dx,al           ;load pointer into WR0

                mov     al,00h          ;disable all
                                        ;interrupts
                out     dx,al           ;issue command
WR3:            mov     al,03h          ;point to W32
                out     dx,al           ;load pointer into WR0

                mov     al,0c1h         ;Rx enable, 8 bits,
                                        ;auto enable off
                out     dx,al           ;issue command
WR4:            mov     al,04h          ;point to WR4
                out     dx,al           ;load pointer into WR0

                mov     al,44h          ;x16 clk, no parity, 1
                                        ;stop bit
                out     dx,al           ;issue command
WR5:            mov     al,05h          ;point to WR5
                out     dx,al           ;load pointer into WR0

                mov     al,6ah          ;Tx enable, 8 Tx
                                        ;bits, RTS enable
                out     dx,al           ;issue command
;
;
;  Receive some data from the other Z-100
;       Test bit D0 of RR0.  If the bit is set, a receive
;character is available.
;

Loop2:          mov     dx,01h          ;control address of
DART
                in      al,dx           ;read RR0
                and     al,01h          ;mask for 1st data bit

                jz      loop2           ;no chara? keep
                                        ;polling
                mov     dx,00h          ;data address of DART
                in      al,dx           ;read character into
                                        ;al
                mov     bl,al           ;store character into
                                        ;bl
;
; Output the character back to the DART.  Check the tx
;buffer to see if it is empty first.  This is done by
;testing bit 3 of RR0.
;
```

```
Loop3:          mov       dx,01h       ;control address of
DART
                in        al,dx        ;read RR0
                and       al,04h       ;mask for bit 3
                jz        loop3        ;wait for the Tx
                                    ;buffer
                mov       dx,00h       ;data address of DART
                mov       al,bl        ;put character into al

                out       dx,al        ;output the character
                jmp       loop2        ;no, output same
                                    ;character
;
;
;
;All do, so exit from program
;
exit3:          mov       ax,4c00h     ;use DOS function 4c
                int       21h          ;to exit from program
;
main            endp
LAB_1           ends
                end       main         ;end assy
```

Appendix D: Test Data

The following test data was collected during the first test of a DART configured as a serial I/O port. The DART was set internally to 9600 baud, one start bit, one stop bit, no parity, and x16 clock. An external clock running at 153.6 Khz was connected to pins 13 and 14 of the DART to provide an internal clock pulse. All equipment was turned on simultaneously and allowed to warm up as the test progressed.

The DART was tested by sending a word (8 bits) to the DART from a Z-100. The DART received the word, then transmitted the word back to the Z-100. The Z-100 checked the received word with the orginal transmitted word to determine if they were equal. Any word found not equal to the transmitted word was counted as one error. Multiple errors within a word, if they occured, were not check and therefore are not accounted for in the following data. Nevertheless, the data given is sufficient to develop an error curve versus time. Once the equipment is sufficiently warm, the error rate drops to zero.

Words transmitted	Approximate Time	Errors Count	Errors
01000	00:00:10	01000	1000
02000	00:00:20	02000	1000
03000	00:00:30	03000	1000
04000	00:00:40	04000	1000
05000	00:00:50	05000	1000
06000	00:00:60	06000	1000
07000	00:00:70	07000	1000
08000	00:00:80	08000	1000
09000	00:00:90	09000	1000
10000	00:01:00	10000	1000
11000	00:01:10	11000	1000
12000	00:01:20	12000	1000
13000	00:01:30	13000	1000
14000	00:01:40	14000	1000
15000	00:01:50	15000	1000
16000	00:01:60	16000	1000
17000	00:01:70	17000	1000
18000	00:01:80	18000	1000
19000	00:01:90	19000	1000
10000	00:02:00	20000	1000
21000	00:02:10	21000	1000
22000	00:02:20	22000	1000
23000	00:02:30	23000	1000
24000	00:02:40	24000	1000
25000	00:02:50	25000	1000
26000	00:02:60	26000	1000
27000	00:02:70	27000	1000
28000	00:02:80	28000	1000
29000	00:02:90	29000	1000
30000	00:03:00	30000	1000
31000	00:03:10	31000	1000
32000	00:03:20	32000	1000

Divide overflow on error count, reset and continue:

33000	00:03:30	01000	1000
34000	00:03:40	02000	1000
35000	00:03:50	03000	1000
36000	00:03:60	04000	1000
37000	00:03:70	05000	1000
38000	00:03:80	06000	1000
39000	00:03:90	07000	1000
40000	00:04:00	08000	1000
41000	00:04:10	09000	1000
42000	00:04:20	10000	1000
43000	00:04:30	11000	1000
44000	00:04:40	12000	1000
45000	00:04:50	13000	1000
46000	00:04:60	14000	1000

Words transmitted	Approximate Time	Errors Count	Errors
47000	00:04:70	15000	1000
48000	00:04:80	16000	1000
49000	00:04:90	17000	1000
50000	00:05:00	18000	1000
51000	00:05:10	19000	1000
52000	00:05:20	20000	1000
53000	00:05:30	21000	1000
54000	00:05:40	22000	1000
55000	00:05:50	23000	1000
56000	00:05:60	24000	1000
57000	00:05:70	25000	1000
58000	00:05:80	26000	1000
59000	00:05:90	27000	1000
60000	00:06:00	28000	1000
61000	00:06:10	29000	1000
62000	00:06:20	30000	1000
63000	00:06:30	31000	1000
64000	00:06:40	32000	1000
Divide overflow on error count, reset and continue:			
65000	00:06:50	00828	0828
66000	00:06:60	01616	0788
67000	00:06:70	02321	0705
68000	00:06:80	03129	0808
69000	00:06:90	03852	0723
70000	00:07:00	04566	0714
71000	00:07:10	05267	0701
72000	00:07:20	05897	0630
73000	00:07:30	06504	0607
74000	00:07:40	07110	0606
75000	00:07:50	07679	0587
76000	00:07:60	08220	0523
77000	00:07:70	08753	0533
78000	00:07:80	09250	0497
79000	00:07:90	09750	0500
80000	00:08:00	10132	0382
81000	00:08:10	10555	0423
82000	00:08:20	10958	0403
83000	00:08:30	11343	0385
84000	00:08:40	11700	0357
85000	00:08:50	12011	0311
86000	00:08:60	12313	0302
87000	00:08:70	12591	0278
88000	00:08:80	12851	0260
89000	00:08:90	13079	0228
90000	00:09:00	13281	0202
91000	00:09:10	13456	0175

Words transmitted	Approximate Time	Errors Count	Errors
92000	00:09:20	13605	0149
93000	00:09:30	13727	0122
94000	00:09:40	13814	0087
95000	00:09:50	13894	0080
96000	00:09:60	13940	0054
97000	00:09:70	13955	0015
98000	00:09:80	14020	0025
99000	00:09:90	14044	0024
100000	00:09:00	14051	0007
101000	00:10:10	14051	0000
102000	00:11:20	14051	0000
103000	00:12:30	14051	0000

An additional five errors were observed as the number of words transmitted increased to 400000 (about 40 minutes after starting). After this time, no additional errors were observed.

Appendix E: BCH Encoder Software

```
;*****************************************************************
;
;      BCH.ASM
;
;      BCH.ASM is a test routine for the BCH encoder.
;BCH.ASM will prompt the user for a multiplier polynomial
;(up to eight bits) which it stores in a one character word.
;
;       NOTA BENE: The BCH encoder must be set to the
;generator polynomial choosen by the user above.
;
;      BCH.ASM then prompts the user for a code word which
;can be any ASCII character.  The code word is multiplied by
;the multiplier polynomial and displayed on the screen.
;Once the results has been displayed, BCH.ASM sends the code
;word through the BCH encoder for encoding.  The encoded
;word is displayed on the screen for easy comparison to the
;multiplied word.
;
;*****************************************************************
;
;  Define constants
;
bell            equ         07h      ;ASCII char for bell
cr              equ         0dh      ;ASCII char for carriage
return
lf              equ         0ah      ;ASCII char for line feed
port1           equ         0000h    ;output port, data
port2           equ         0001h    ;output port, control
;
;
;*****************************************************************
;
stack_area      segment     stack          ;define the stack area
                db          40 dup(?)      ;set aside room for
stack
stack_area      ends
;
;
data_area       segment                    ;define data area
msg1      db        cr,lf
          db        'Enter the generator polynmial (-30h):',
          db        cr,lf,'$'
msg2      db        cr,lf,lf
          db        'Enter the code word:',cr,lf,'$'
msg3      db        '1','$'
msg4      db        '0','$'
msg5      db        cr,lf,'$'
msg6      db        'Enter the value of n-k:','$'
```

```
GEN       db        00h
CODE      dw        000Ch
MWORD     dw        0000h
NK        db        00h
data_area      ends
;
;*****************************************************************
;
;
LAB_1          segment          ;define segment area for
                          ;program
main           proc      far
               assume    cs:LAB_1, ds:data_area,
ss:stack_area
;
;set all register to zero (makes it easier to debug)
;
begin:         sub       ax,ax                ;set ax to zero
               sub       bx,bx                ;set bx to zero
               sub       cx,cx                ;set cx to zero
               sub       dx,dx                ;set dx to zero
;
;set DS reg to point to the current data area
;
          mov       ax,data_area    ;point to data_area
          mov       ds,ax           ;DS now points to
data_area
;
;
;  Initialize the DART to x16 clock, 1 stop bit, 8 Tx/Rx ;
bits, no parity, and no interrupts (polled mode).
;
;      AL contains the control byte to the DART
;      DX contains the control byte address

          push      ax              ;save ax
          push      dx              ;save dx
          mov       dx,port2        ;output port address
          mov       al,18h          ;channel reset command
          out       dx,al           ;issue command
          out       dx,al           ;again, just to be sure
WR1:      mov       al,01h          ;point to WR1
          out       dx,al           ;load pointer into WR0
          mov       al,00h          ;disable all interrupts
          out       dx,al           ;issue command
WR2:      mov       al,02h          ;point to WR2
          out       dx,al           ;load pointer into WR0
          mov       al,00h          ;disable all interrupts
          out       dx,al           ;issue command
WR3:      mov       al,03h          ;point to WR3
          out       dx,al           ;load pointer into WR0
          mov       al,0c1h         ;Rx enable, 8 bits,
```

```
                                    ;auto enable off
        out     dx,al               ;issue command
WR4:    mov     al,04h              ;point to WR4
        out     dx,al               ;load pointer into WR0
        mov     al,44h              ;x16 clk, no parity, 1 stop
bit
        out     dx,al               ;issue command
WR5:    mov     al,05h              ;point to WR5
        out     dx,al               ;load pointer into WR0
        mov     al,6ah              ;Tx enable, 8 Tx bits,RTS
                                ;enable
        out     dx,al               ;issue command
;
;  Disable the DART receiver and transmitter, so that
;  the BCH encoder can be used without affecting ;
transmissions.
;
        mov     al,03h              ;point to WR3
        out     dx,al               ;load pointer into WR0
        mov     al,0c1h             ;Rx disabled, 8 bits,
                                    ;auto enable off

        mov     al,05h              ;point to WR5
        out     dx,al               ;load pointer into WR0
        mov     al,6ah              ;Tx disabled, 8 Tx bits,
        out     dx,al               ;RTS disabled

;
;  Prompt for the generator polynomial:
;
        mov      dx,offset msg1     ;load dx with add msg1
        mov      ah,09h
        int      21h

        mov      ah,01h             ;Read char and echo
        int      21h                ;Char in al
        sub      al,30h             ;convert to binary
        mov      GEN,al             ;store char in bl

        mov      dx,offset msg5     ;to cr and lf
        mov      ah,09h
        int      21h
;
;   Prompt for value of n-k:
;
        mov      dx,offset msg6     ;prompt for n-k
        mov      ah,09h
        int      21h

        mov      ah,01h             ;read char and echo
        int      21h                ;n-k in al
        sub      al,30h             ;convert to binary
```

```
        mov     NK,al           ;store n-k in NK

        mov     dx,offset msg5  ;to cr and lf
        mov     ah,09h
        int     21h
;
;  Prompt for the code word:
;
Loop1:  mov     dx,offset msg2  ;load dx with add msg2
        mov     ah,09h
        int     21h

        mov     ah,01h          ;Read char and echo
        int     21h             ;Char in al
        sub     bx,bx           ;clear bx
        mov     bl,al           ;put code word in bl
        mov     CODE,bx         ;store code word

        mov     dx,offset msg5  ;load cr and lf
        mov     ah,09h
        int     21h

;
;  Multilply code word with generator polylnomial and ;
display  the result (Note, this is not binary ;
multiplication):
;
        sub     bx,bx           ;clear bx
        sub     ax,ax           ;clear ax (holds ans)
        mov     dx,CODE         ;code word in dx
        mov     bh,GEN          ;put gen poly in bh
        mov     cx,0008h        ;count 8 bits


ContM:  shr     bx,1            ;shift bit into bx
        and     bl,80h          ;mask for high bit
        cmp     bl,00h          ;is bit 0 or 1
        je      Skip            ;yes, skip xor
        xor     ax,dx           ;answer in ax
Skip:   shl     dx,1            ;get ready for next xor

        dec     cx              ;decrease count
        cmp     cx,0000h        ;all done?
        jg      ContM           ;no, continue
                        ;multipling
        mov     bx,ax           ;answer in bx
        call    Prnt

;
;   Now generate the encoded word using the BCH encoder
;
```

```
        mov     bx,CODE         ;bx has code word
        mov     cl,NK           ;cl loaded with n-k
        shl     bx,cl           ;add zero padding
        mov     CODE,bx         ;store padded code

        mov     dx,8000h        ;to clear the encoder
        mov     al,00h
        out     dx,al

        mov     cx,000fh        ;cx counts 15 bits
LOOP3:  mov     bx,CODE         ;bx has code char
        mov     dx,4000h        ;to clock data into
                                ;BCH encoder
        mov     al,bh           ;mov bh into al
        out     dx,al           ;output high bit
        shl     bx,1            ;shift bx left 1 bit
        mov     CODE,bx         ;store code word

;  Get the encoded word

LOOP4:  mov     dx,0c000h       ;to input a bit
        sub     bx,bx           ;clear bx
        in      al,dx           ;al has low order bit
        and     al,01h          ;strip off low bit
        mov     bl,al           ;bl has low order bit
        mov     ax,MWORD        ;al has encoded word
        or      ax,bx           ;append new bit

        mov     bx,ax           ;Prnt prints bx
        cmp     cx,0001h        ;all done?
        je      GoPrnt          ;yes, Print result

        shl     ax,1            ;shift al left 1 bit
        mov     MWORD,ax        ;store result in MWORD
        dec     cx              ;cx holds count
        jmp     LOOP3           ;finish getting result
;
GoPrnt: call    Prnt
        sub     ax,ax           ;clear ax
        mov     MWORD,ax        ;clear MWORD
        jmp     Loop1


;****************************************************************
;
;  Subroutine to print the result
;  Word to be printed must be in bx
;
;****************************************************************

Prnt    proc    near
        mov     cx,0010h        ;cx counts 16 bits
```

```
Loop2:  mov     ax,bx           ;put answer in ax
        and     ax,8000h        ;mask for high order
                                ;bit
        cmp     ax,0000h        ;is bit a zero?
        je      Zero
        mov     dx,offset msg3  ;output a "1"
        mov     ah,09h
        int     21h
        jmp     Cont

Zero:   mov     dx,offset msg4  ;output a "0"
        mov     ah,09h
        int     21h

Cont:   dec     cx              ;decrease counter
        cmp     cx,0000h        ;all done?
        je      Go              ;yes, exit
        shl     bx,1            ;get next bit and
        jmp     Loop2           ;continue

Go:     mov     dx,offset msg5  ;cr and lf
        mov     ah,09h
        int     21h
        ret
Prnt    endp

;*************************************************************
;
;All do, so exit from program
;
Exit:   mov     ax,4c00h    ;use DOS function 4c
        int     21h         ;to exit from program
;
main    endp
LAB_1   ends
        end     main        ;end assy
```

Appendix F: BCH Decoder Single Character Software

```
;*****************************************************************
;
;     DECODE.ASM
;
;     DECODE.ASM is a test routine for the BCH decoder.
;DECODE.ASM will prompt the user for a multiplier polynomial
;(up to eight bits) which it stores in a one character word.
;
;     NOTA BENE: The BCH decoder must be set to the
;generator polynomial choosen by the user above.
;
;     DECODE.ASM then prompts the user for a code word which
;can be any ASCII character.  The code word is multiplied by
;the multiplier polynomial and displayed on the screen.
;Once the results has been displayed, BCH.ASM sends the
;encoded word through the BCH decoder for decoding.  The
;decoded word is displayed on the screen for easy comparison
;to the original code word.
;
;*****************************************************************
;
;  Define constants
;
bell            equ       07h     ;ASCII char for bell
cr              equ       0dh     ;ASCII char for carriage
                                  ;return
lf              equ       0ah     ;ASCII char for line feed
port1           equ       0000h   ;output port, data
port2           equ       0001h   ;output port, control
;
;
;*****************************************************************
;
stack_area      segment   stack         ;define the stack area
                db        40 dup(?)     ;set aside room for
stack
stack_area      ends
;
;
data_area       segment                 ;define data area
msg1      db        cr,lf
          db        'Enter the generator polynmial (-30h):',
          db        cr,lf,'$'
msg2      db        cr,lf,lf
          db        'Enter the code word:',cr,lf,'$'
msg3      db        '1','$'
msg4      db        '0','$'
msg5      db        cr,lf,'$'
msg6      db        'Enter the value of n-k:','$'
```

```
msg7      db        'Enter the value of k:','$'
GEN       db        00h
CODE      dw        0000h
MWORD     dw        0000h
NK        db        00h
K         db        00h
RESULT    db        00h
data_area      ends
;
;*****************************************************************
;
;
LAB_1          segment          ;define segment area for
                           ;program
main           proc       far
               assume     cs:LAB_1, ds:data_area,
ss:stack_area
;
;set all register to zero (makes it easier to debug)
;
begin:         sub        ax,ax                ;set ax to zero
               sub        bx,bx                ;set bx to zero
               sub        cx,cx                ;set cx to zero
               sub        dx,dx                ;set dx to zero
;
;set DS reg to point to the current data area
;
          mov        ax,data_area    ;point to data_area
          mov        ds,ax           ;DS now points to
data_area
;
;
;  Initialize the DART to x16 clock, 1 stop bit, 8 Tx/Rx ;
bits, no parity, and no interrupts (polled mode).
;
;       AL contains the control byte to the DART
;       DX contains the control byte address

          push       ax              ;save ax
          push       dx              ;save dx
          mov        dx,port2        ;output port address
          mov        al,18h          ;channel reset command
          out        dx,al           ;issue command
          out        dx,al           ;again, just to be sure
WR1:      mov        al,01h          ;point to WR1
          out        dx,al           ;load pointer into WR0
          mov        al,00h          ;disable all interrupts
          out        dx,al           ;issue command
WR2:      mov        al,02h          ;point to WR2
          out        dx,al           ;load pointer into WR0
          mov        al,00h          ;disable all interrupts
          out        dx,al           ;issue command
```

```
WR3:    mov     al,03h          ;point to WR3
        out     dx,al           ;load pointer into WR0
        mov     al,0c1h         ;Rx enable, 8 bits,
                                ;auto enable off
        out     dx,al           ;issua command
WR4:    mov     al,04h          ;point to WR4
        out     dx,al           ;load pointer into WR0
        mov     al,44h          ;x16 clk, no parity, 1 stop
                                ;bit
        out     dx,al           ;issue command
WR5:    mov     al,05h          ;point to WR5
        out     dx,al           ;load pointer into WR0
        mov     al,6ah          ;Tx enable, 8 Tx bits,RTS
                            ;enable
        out     dx,al           ;issue command
;
;  Disable the DART receiver and transmitter, so that
;  the BCH decoder can be used without affecting ;
transmissions.
;
        mov     al,03h          ;point to WR3
        out     dx,al           ;load pointer into WR0
        mov     al,0c1h         ;Rx disabled, 8 bits,
                                ;auto enable off

        mov     al,05h          ;point to WR5
        out     dx,al           ;load pointer into WR0
        mov     al,6ah          ;Tx disabled, 8 Tx bits,
        out     dx,al           ;RTS disabled

;
;  Prompt for the generator polynomial:
;
        mov      dx,offset msg1   ;load dx with add msg1
        mov      ah,09h
        int      21h

        mov      ah,01h           ;Read char and echo
        int      21h              ;Char in al
        sub      al,30h           ;convert to binary
        mov      GEN,al           ;store char in bl

        mov      dx,offset msg5   ;to cr and lf
        mov      ah,09h
        int      21h
;
;   Prompt for value of n-k:
;
        mov      dx,offset msg6   ;prompt for n-k
        mov      ah,09h
        int      21h
```

```
        mov     ah,01h          ;read char and echo
        int     21h             ;n-k in al
        sub     al,30h          ;convert to binary
        mov     NK,al           ;store n-k in NK

        mov     dx,offset msg5  ;to cr and lf
        mov     ah,09h
        int     21h

;
;    Prompt for value of k:
;
        mov     dx,offset msg7   ;prompt for k
        mov     ah,09h
        int     21h

        mov     ah,01h           ;read char and echo
        int     21h              ;k in al
        sub     al,30h           ;convert to binary
        mov     K,al             ;store k in K

        mov     dx,offset msg5   ;to cr and lf
        mov     ah,09h
        int     21h

;
;  Prompt for the code word:
;
Loop1:  mov     dx,offset msg2  ;load dx with add msg2
        mov     ah,09h
        int     21h

        mov     ah,01h          ;Read char and echo
        int     21h             ;Char in al
        sub     bx,bx           ;clear bx
        mov     bl,al           ;put code word in bl
        mov     CODE,bx         ;store code word

        mov     dx,offset msg5  ;load cr and lf
        mov     ah,09h
        int     21h

        mov     bx,CODE
        call    Prnt

;
;  Multilply code word with generator polylnomial and ;
display the result (Note, this is not binary ;
multiplication):
;
        sub     bx,bx           ;clear bx
        sub     ax,ax           ;clear ax (holds ans)
```

```
        mov     dx,CODE         ;code word in dx
        mov     bh,GEN          ;put gen poly in bh
        mov     cx,0008h        ;count 8 bits

ContM:  shr     bx,1            ;shift bit into bx
        and     bl,80h          ;mask for high bit
        cmp     bl,00h          ;is bit 0 or 1
        je      Skip            ;yes, skip xor
        xor     ax,dx           ;answer in ax
Skip:   shl     dx,1            ;get ready for next xor

        dec     cx              ;decrease count
        cmp     cx,0000h        ;all done?
        jg      ContM           ;no, continue
                        ;multipling

        mov     MWORD,ax        ;store answer in MWORD
        mov     bx,ax           ;answer in bx
        call    Prnt

;
;   Now decode the encoded word using the BCH decoder
;
        mov     dx,0000h        ;to clear the decoder
        mov     al,00h
        out     dx,al


;
;   Clock in n-k data bits
;
        mov     bx,MWORD        ;encoded word in bx

BHigh:  mov     al,NK           ;al has n-k
        mov     dl,K            ;dl has k
        add     al,dl           ;al has n
        mov     cl,16d          ;cl has 16
        sub     cl,al           ;cl has # of zeros
        shl     bx,cl           ;shift out zeros

        sub     cx,cx           ;clear cx
        mov     cl,NK           ;cx counts n-k bits
        mov     dx,4000h        ;to clock data into
                                ;BCH decoder

LOOP3:  mov     al,bh           ;high order bit in A7
        out     dx,al           ;output high bit
        shl     bx,1            ;shift bx left once
        dec     cx              ;count down n-k
        cmp     cx,0000h        ;all done?
```

```
        je      DEC             ;yes, finish decoding
        jmp     LOOP3           ;finish loading bits

;
;  Next k bits are the decoded word
;
DEC:    mov     cl,K            ;to count k bits
        mov     RESULT,00h      ;clear result
LOOP4:  mov     dx,0c000h       ;input a bit into CPU
        in      al,dx           ;bit is in A0
        and     al,01h          ;mask for D0 (A0)
        mov     dl,RESULT       ;decoder word in dl
        shl     dl,1            ;shift deocded word
                                ;left 1
        xor     dl,al           ;add bit to decoded
                                ;word
        mov     RESULT,dl       ;store result

        mov     dx,4000h        ;to output a bit
        mov     al,bh           ;high order bit in A7
        out     dx,al           ;output the bit
        shl     bx,1            ;shift encoded word
                                ;left 1

        dec     cl              ;decrease count
        cmp     cl,00h          ;all done?
        je      OUT             ;yes, go to OUT
        jmp     LOOP4           ;finish decoding

OUT:    sub     bx,bx           ;clear bx
        mov     bl,RESULT       ;place result in bl
        call    Prnt            ;Print result
;
;   Output the character to the screen
;
        mov     dl,RESULT       ;character in dl
        mov     ah,02h          ;DOS call to prnt char
        int     21h
        jmp     Loop1


;****************************************************************
;
;  Subroutine to print the result
;  Word to be printed must be in bx
;
;****************************************************************

Prnt    proc    near
        mov     cx,0010h        ;cx counts 16 bits

Loop2:  mov     ax,bx           ;put answer in ax
```

```
          and       ax,8000h          ;mask for high order
                                    ;bit
          cmp       ax,0000h          ;is bit a zero?
          je        Zero
          mov       dx,offset msg3    ;output a "1"
          mov       ah,09h
          int       21h
          jmp       Cont

Zero:     mov       dx,offset msg4    ;output a "0"
          mov       ah,09h
          int       21h

Cont:     dec       cx                ;decrease counter
          cmp       cx,0000h          ;all done?
          je        Go                ;yes, exit
          shl       bx,1              ;get next bit and
          jmp       Loop2             ;continue

Go:       mov       dx,offset msg5    ;cr and lf
          mov       ah,09h
          int       21h
          ret
Prnt      endp

;************************************************************

;
;All do, so exit from program
;
Exit:     mov       ax,4c00h     ;use DOS function 4c
          int       21h          ;to exit from program
;
main      endp
LAB_1     ends
          end       main         ;end assy
^Z
```

Appendix G: BCH Decoder Syndrome Software

```
;*****************************************************************
;
;     DEC1.ASM generates the syndromes for the (15,7) two
;error correcting BCH code using the generator polynomial
;x^8 + x^7 + x^6  + x^4 + 1.  All error polynomials and
;syndromes are generated in software and sent to the printer
;for a hard copy display of the results.
;
;*****************************************************************
;
;  Define constants
;
bell            equ         07h     ;ASCII char for bell
cr              equ         0dh     ;ASCII char for carriage
                                    ;return
lf              equ         0ah     ;ASCII char for line feed
port1           equ         0000h   ;output port, data
port2           equ         0001h   ;output port, control
;
;
;*****************************************************************
;
stack_area      segment     stack           ;define the stack area
                db          40 dup(?)       ;set aside room for
stack
stack_area      ends
;
;
data_area       segment                     ;define data area
GEN         dw          01d1h
CODE        dw          0000h
MWORD       dw          0000h
NK          db          08h
K           db          07h
N           db          0fh
T           db          02h
Base        dw          0001h
Drv         dw          0002h
RESULT      db          00h
MSK         dw          0000h
Count       db          00h
LIST        db      300 dup ('     ')
Extr        db       10 dup ('     ')
data_area       ends
;
;*****************************************************************
;
;
LAB_1           segment           ;define segment area for
```

```
program
main            proc        far
                assume      cs:LAB_1, ds:data_area,
            ss:stack_area
;
;set all register to zero (makes it easier to debug)
;
                sub         ax,ax                   ;set ax to zero
                sub         bx,bx                   ;set bx to zero
                sub         cx,cx                   ;set cx to zero
                sub         dx,dx                   ;set dx to zero
;
;set DS reg to point to the current data area
;
          mov       ax,data_area    ;point to data_area
          mov       ds,ax           ;DS now points to
data_area

          call      Generate
          call      LST
          jmp       Exit
;
;*****************************************************************
;
;   Generate is a subroutine that generates the syndroms for
;all possible error conditions allowed by the given code
;structure.
;
;   Note that cx contains the length of the encoded word
;         and dx contains the number of correctable errors
;
;*****************************************************************
;
Generate    proc    near
;
;  Generate single error polynomials
;
         push       ax
         push       bx
         push       cx
         push       dx
;
;  Clear LIST to all Zeros
;
         sub        bx,bx               ;bx has all zeros
         mov        SI,offset LIST      ;SI points to LIST
         mov        cx,300d             ;cx has number of
                                        ;records

Rpt:     mov        [SI],bx             ;output zeros
         add        SI,02h              ;point to next record
         mov        [SI],bx             ;output zeros
```

```
        add     SI,02h          ;point to next record
        dec     cx
        cmp     cx,00h          ;all done?
        je      Sta
        jmp     Rpt

;  Compute single errors:
;
Sta:    mov     SI,offset LIST  ;SI points to LIST
        mov     bx,0001h        ;bx used to generate
                                ;error
                                ;polynomials
        mov     cl,N            ;cx has length of
                                ;polynomial
Goop1:  mov     [SI],bx         ;store bx in LIST
        call    SYNDROME        ;calculate the syndrom
        add     SI,02h          ;SI points to next pos
        shl     bx,1            ;generate next error
                                ;poly
        dec     cl              ;count down number of
                                ;errors
        cmp     cl,01h          ;all done?
        je      Gext            ;do double errors
        jmp     Goop1           ;no, finish single
                                ;errors

; Compute double errors if number of errors = 2
;
Gext:   mov     cl,T            ;number of errors in cl
        cmp     cl,02h          ;double errors
                                ;indicated?
        jne     Gxit            ;no, skip double errors

        mov     cl,N            ;cx has length of
                                ;polynomial
        mov     dl,N            ;dx has length of
                                ;polynmoial
        dec     dl              ;dx has N-1
        mov     Count,dl        ;store N-1 in Count
        mov     bx,Base         ;bx used to generate
                                ;error
                                ;plynomials
        mov     ax,drv          ;ax has 02h

Goop2:  xor     bx,ax           ;bx contains double
                                ;error
        mov     [SI],bx         ;store bx in LIST
        call    SYNDROME        ;calculate the syndrome
        add     SI,02h          ;SI points to next pos
        mov     bx,Base         ;reinitialize bx
        shl     ax,1            ;shift ax left once
        dec     dl              ;to count shifts
```

```
        cmp     dl,00h          ;all done yet
        je      Chang           ;yes change bx and ax
        jmp     Goop2           ;no continue

Chang:  dec     cl              ;decrease length by one
        cmp     cl,01h          ;all combinations done
                                ;yet?
        je      Gxit            ;all done, exit
        mov     bx,Base         ;load bx with Base
        shl     bx,1            ;shift base left once
        mov     Base,bx         ;store bx in Base
        mov     ax,Drv          ;ax has driver
        shl     ax,1            ;shift ax left once
        mov     Drv,ax          ;store new driver in Drv

        mov     dl,Count        ;put Count in dl
        dec     dl
        mov     Count,dl        ;store Count-1 in Count
        jmp     Goop2

Gxit:   pop     dx
        pop     cx
        pop     bx
        pop     ax

        ret
Generate    endp

;*****************************************************************
;
;     SYNDROME is a subroutine to calculate the syndrome of
;an error polynomial given the generator polynomial.
;
;     Note:  The error polynomial must be in bx, the
;     generator polynomial is stored as a global variable
;     in GEN.  SI has been initialzed to point to LIST by
;     subroutine Generate.
;
;*****************************************************************

SYNDROME   proc    near

         push       ax
         push       bx
         push       cx
         push       dx

; Place generator polynomial in ax to use as the divisor
; and intialize ax by shifting the high order bit as far
; left as possible

         mov     dx,8000h        ;mask for high order bit
```

```
        mov     ax,GEN          ;ax load with gen poly
Soop1:  and     dx,ax           ;test for high order bit
        cmp     dx,8000h        ;high order bit set?
        je      Sext            ;yes, procede
        shl     ax,1            ;no, shift ax left once
        mov     dx,8000h        ;reinitialize dx
        jmp     Soop1

; Ready to calculate the Syndrome

Sext:   mov     cl,16d          ;number of bits in a
                                ;register
        sub     cl,NK           ;cl has # of shifts needed
                                ;to divide error polynmial
        mov     dx,8000h        ;to mask high bit
        mov     MSK,dx          ;store mask in MSK
Soop2:  and     dx,bx           ;is bit high?
        cmp     dx,0000h        ;test for bit high.
        je      Shft            ;no, shift registers
        xor     bx,ax           ;xor (divide by steps)
                                ;bx/ax
Shft:   cmp     cl,01h          ;all done?
        je      Stor            ;store syndrome
        mov     dx,MSK          ;load dx with mask
        shr     dx,1            ;shift mask right once
        mov     MSK,dx          ;store new mask in MSK
        shr     ax,1            ;shift gen poly right once
        dec     cl              ;decrease count by one
        jmp     Soop2           ;repeat the process

; Store the syndrome

Stor:   add     SI,02h          ;point to storage location
        mov     [SI],bx         ;syndrome (remainder)
                                ;stored

; Restore registers

        pop     dx
        pop     cx
        pop     bx
        pop     ax
        ret

SYNDROME    endp

;*****************************************************************
;
;  Subroutine to print the result
;  Word to be printed must be in bx
;
;*****************************************************************
```

```
Prnt    proc    near

        push    ax
        push    bx
        push    cx
        push    dx

        mov     cx,0010h        ;cx counts 16 bits

Loop2:  mov     ax,bx           ;put answer in ax
        and     ax,8000h        ;mask for high order
                                ;bit
        cmp     ax,0000h        ;is bit a zero?
        je      Zero
        mov     dl,31h          ;output a 1
        mov     ah,05h
        int     21h
        jmp     Cont

Zero:   mov     dl,30h          ;output a 0
        mov     ah,05h
        int     21h

Cont:   dec     cx              ;decrease counter
        cmp     cx,0000h        ;all done?
        je      Go              ;yes, exit
        shl     bx,1            ;get next bit and
        jmp     Loop2           ;continue

Go:     pop     dx
        pop     cx
        pop     bx
        pop     ax

        ret
Prnt    endp

;*************************************************************
;
;   CRLF is a subroutine to output a carriage return
;and a line feed.
;
;*************************************************************

CRLF    Proc    near

        mov     dl,0dh      ;cr
        mov     ah,05h
        int     21h

        mov     dl,0ah      ;lf
        mov     ah,05h
```

```
        int     21h

        ret
CRLF    endp

;*****************************************************************
;
;    BLNK is a subroutine to output four blanks
;
;*****************************************************************

BLNK    proc    near

        mov     dx,20h          ;four blanks
        mov     ah,05h
        int     21h

        ret
BLNK    endp

;*****************************************************************

;
;    LST is a subroutine to list the contents of memory in
;a binary format;
;
;    Note: will only list the contents of memory stored in
;memory location LIST
;
;*****************************************************************


LST     proc    near

        mov     SI,offset LIST  ;SI points to LIST

;Calculate length of list to display

        mov     al,T            ;ax has number of errors
        cmp     al,02h          ;two errors?
        je      Two
        sub     cx,cx           ;clear cx
        mov     cl,N            ;n single errors
        jmp     Show

Two:    sub     cx,cx           ;clear cx
        sub     bx,bx           ;clear bx
        mov     cl,N            ;length of polynomial
        mov     bl,N            ;length of polynomial
        dec     bx              ;bx has N-1
Agn:    add     cx,bx           ;cx has # of error polys
        cmp     bx,000h         ;all done?
```

```
        je      Show            ;yes, print the results
        dec     bx              ;next number to add
        jmp     Agn

;Display the list

Show:   mov     bx,[SI]         ;polynomial to be
                                ;displayed
        call    Prnt            ;print the results
        call    BLNK            ;add some blanks
        inc     SI
        inc     SI              ;point to next record
        mov     bx,[SI]         ;load bx with next record
        call    Prnt            ;print the next record
        call    CRLF            ;Cr and Lf
        inc     SI
        inc     SI              ;point to next record
        dec     cx              ;decrement counter
        cmp     cx,00h          ;all done?
        je      DN              ;yes, exit
        jmp     Show            ;no, get next record

DN:     ret
LST     endp
;***************************************************************

;All do, so exit from program
;
Exit:   mov     ax,4c00h        ;use DOS function 4c
        int     21h             ;to exit from program
;
main    endp
LAB_1   ends
        end     main            ;end assy
^Z

;***************************************************************
;
;   DEC2.ASM tests the BCH decoder by generating all the
;sydromes for the (15,7) BCH two error correcting code using
;the generator polynomial x^8 + x^7 + x^6 + x^4 + 1. Each
;possible error polynomial along with the generated syndrome
;is sent  to the printer for a hard copy of the results.
;The results of this program are to be compared witht the
;results of DEC1.ASM (which generates the correct syndromes
;in software).  When the results of DEC2.ASM match the
;results of DEC1.ASM, the BCH decoder is working properly.
;
;***************************************************************
;
;  Define constants
;
```

```
bell            equ         07h     ;ASCII char for bell
cr              equ         0dh     ;ASCII char for carriage
return
lf              equ         0ah     ;ASCII char for line feed
port1           equ         0000h   ;output port, data
port2           equ         0001h   ;output port, control
;
;***************************************************************
;
stack_area      segment     stack           ;define the stack area
                db          40 dup(?)       ;set aside room for
stack
stack_area      ends
;
;
data_area       segment                     ;define data area
GEN       dw        01d1h
CODE      dw        0000h
MWORD     dw        0000h
NK        db        08h
K         db        07h
N         db        0fh
T         db        02h
Base      dw        0001h
Drv       dw        0002h
RESULT    db        00h
MSK       dw        0000h
Count     db        00h
LIST      db    300 dup ('     ')
Extr      db     10 dup ('     ')
data_area       ends
;
;***************************************************************
;
;
LAB_1           segment         ;define segment area for
                                ;program
main            proc        far
                assume      cs:LAB_1, ds:data_area,
ss:stack_area
;
;set all register to zero (makes it easier to debug)
;
                sub         ax,ax               ;set ax to zero
                sub         bx,bx               ;set bx to zero
                sub         cx,cx               ;set cx to zero
                sub         dx,dx               ;set dx to zero
;
;set DS reg to point to the current data area
;
          mov       ax,data_area    ;point to data_area
          mov       ds,ax           ;DS now points to
```

```
                        ;data_area

        call    Generate
        call    LST
        jmp     Exit
;
;*****************************************************************
;
;   Generate is a subroutine that generates the syndroms for
;all possible error conditions allowed by the given code
;structure.
;
;   Note that cx contains the length of the encoded word
;         and dx contains the number of correctable errors
;
;*****************************************************************
;
Generate   proc   near
;
;  Generate single error polynomials
;
        push    ax
        push    bx
        push    cx
        push    dx
;
;  Clear LIST to all Zeros
;
        sub     bx,bx           ;bx has all zeros
        mov     SI,offset LIST  ;SI points to LIST
        mov     cx,300d         ;cx has number of
                                ;records

Rpt:    mov     [SI],bx         ;output zeros
        add     SI,02h          ;point to next record
        mov     [SI],bx         ;output zeros
        add     SI,02h          ;point to next record
        dec     cx
        cmp     cx,00h          ;all done?
        je      Sta
        jmp     Rpt

;  Compute single errors:
;
Sta:    mov     SI,offset LIST  ;SI points to LIST
        mov     bx,0001h        ;bx used to generate
                                ;error
                                ;polynomials
        mov     cl,N            ;cx has length of
                                ;polynomial
Goop1:  mov     [SI],bx         ;store bx in LIST
        call    DECODE          ;calculate the syndrom
```

```
        add     SI,02h          ;SI points to next pos
        shl     bx,1            ;generate next error
                                ;poly
        dec     cl              ;count down number of
                                ;errors
        cmp     cl,01h          ;all done?
        je      Gext            ;do double errors
        jmp     Goop1           ;no, finish single
                                ;errors

; Compute double errors if number of errors = 2
;
Gext:   mov     cl,T            ;number of errors in cl
        cmp     cl,02h          ;double errors
                                ;indicated?
        jne     Gxit            ;no, skip double errors

        mov     cl,N            ;cx has length of
                                ;polynomial
        mov     dl,N            ;dx has length of
                                ;polynmoial
        dec     dl              ;dx has N-1
        mov     Count,dl        ;store N-1 in Count
        mov     bx,Base         ;bx used to generate
                                ;error
                                ;plynomials
        mov     ax,drv          ;ax has 02h

Goop2:  xor     bx,ax           ;bx contains double
                                ;error
        mov     [SI],bx         ;store bx in LIST
        call    DECODE          ;calculate the syndrome
        add     SI,02h          ;SI points to next pos
        mov     bx,Base         ;reinitialize bx
        shl     ax,1            ;shift ax left once
        dec     dl              ;to count shifts
        cmp     dl,00h          ;all done yet
        je      Chang           ;yes change bx and ax
        jmp     Goop2           ;no continue

Chang:  dec     cl              ;decrease length by one
        cmp     cl,01h          ;all combinations done
                                ;yet?
        je      Gxit            ;all done, exit
        mov     bx,Base         ;load bx with Base
        shl     bx,1            ;shift base left once
        mov     Base,bx         ;store bx in Base
        mov     ax,Drv          ;ax has driver
        shl     ax,1            ;shift ax left once
        mov     Drv,ax          ;store new driver in Drv

        mov     dl,Count        ;put Count in dl
```

```
        dec         dl
        mov         Count,dl          ;store Count-1 in Count
        jmp         Goop2

Gxit:   pop         dx
        pop         cx
        pop         bx
        pop         ax

        ret
Generate    endp

;*****************************************************************
;
;   DECODE will find the syndrom for each error polynmial
;using the BCH decoder.
;
;   The error polynomial must be in bx when DECODE is ;
called.  SI points to LIST (initialized by the calling ;
routine)
;
;*****************************************************************

DECODE  proc        near

        push        ax
        push        bx
        push        cx
        push        dx

        mov         dx,0000h          ;to clear the decoder
        mov         al,00h
        out         dx,al
;
;   Clock in n-k data bits
;
BHigh:  mov         al,NK             ;al has n-k
        mov         dl,K              ;dl has k
        add         al,dl             ;al has n
        mov         cl,16d            ;cl has 16
        sub         cl,al             ;cl has # of zeros
        shl         bx,cl             ;shift out zeros

        sub         cx,cx             ;clear cx
        mov         cl,NK             ;cx counts n-k bits
        mov         dx,4000h          ;to clock data into
                                      ;BCH decoder

DOOP3:  mov         al,bh             ;high order bit in A7
        out         dx,al             ;output high bit
        shl         bx,1              ;shift bx left once
        dec         cx                ;count down n-k
```

```
        cmp     cx,0000h        ;all done?
        je      DEC             ;yes, finish decoding
        jmp     DOOP3           ;finish loading bits

;
;  Next k bits are the decoded word
;
DEC:    mov     cl,K            ;to count k bits
        mov     RESULT,00h      ;clear result
DOOP4:  mov     dx,0c000h       ;input a bit into CPU
        in      al,dx           ;bit is in A0
        and     al,01h          ;mask for D0 (A0)
        mov     dl,RESULT       ;decoder word in dl
        shl     dl,1            ;shift deocded word
                                ;left 1
        xor     dl,al           ;add bit to decoded
                                ;word
        mov     RESULT,dl       ;store result

        dec     cl              ;decrease count
        cmp     cl,00h          ;all done?
        je      SYND            ;yes, get syndrome

        mov     dx,4000h        ;to output a bit
        mov     al,bh           ;high order bit in A7
        out     dx,al           ;output the bit
        shl     bx,1            ;shift encoded word
                                ;left 1
        jmp     DOOP4           ;finish decoding

;  Syndrome (remainder) is in the n-k registers forming
;the generator polynmial

SYND:   mov     dx,8000h        ;output a bit, gate
                                ;open
        mov     al,bh           ;to output last bit
        mov     cl,08h          ;to count NK bits

;  One more time with gate closed.
        mov     dx,4000h        ;output, gate open
        out     dx,al           ;clock in last bit
        mov     al,00h          ;output zeros now
        sub     bx,bx           ;make sure bx clear

;  Get syndrome bits and continue to clock in bits with the
;gate open.

DOOP:   mov     dx,0c000h       ;to input a bit
        in      al,dx           ;bit in A0
        and     al,01h          ;mask for low bit
        xor     bl,al           ;bx has syndrome
        dec     cl              ;decrease bit count
```

```
            cmp         cl,00h          ;all done?
            je          Dstr            ;store the syndrome
            shl         bx,1            ;ready for next bit
            mov         dx,8000h        ;output a bit, gate
                                        ;open
            mov         al,00h          ;to output a zero
            out         dx,al           ;output, gate closed
            jmp         DOOP            ;repeat

Dstr:       add         SI,02h          ;SI points to loc in
                                        ;LIST
            mov         [SI],bx         ;store syndrome

;  Restore registers

            pop         dx
            pop         cx
            pop         bx
            pop         ax
            ret

DECODE      endp

;*****************************************************************
;
;  Subroutine to print the result
;  Word to be printed must be in bx
;
;*****************************************************************

Prnt        proc        near

            push        ax
            push        bx
            push        cx
            push        dx

            mov         cx,0010h        ;cx counts 16 bits

Loop2:      mov         ax,bx           ;put answer in ax
            and         ax,8000h        ;mask for high order
                                        ;bit
            cmp         ax,0000h        ;is bit a zero?
            je          Zero
            mov         dl,31h          ;output a 1
            mov         ah,05h
            int         21h
            jmp         Cont

Zero:       mov         dl,30h          ;output a 0
            mov         ah,05h
            int         21h
```

```
Cont:   dec     cx              ;decrease counter
        cmp     cx,0000h        ;all done?
        je      Go              ;yes, exit
        shl     bx,1            ;get next bit and
        jmp     Loop2           ;continue

Go:     pop     dx
        pop     cx
        pop     bx
        pop     ax

        ret
Prnt    endp

;************************************************************
;
;   CRLF is a subroutine to output a carriage return
;and a line feed.
;
;************************************************************

CRLF    Proc    near

        mov     dl,0dh      ;cr
        mov     ah,05h
        int     21h

        mov     dl,0ah      ;lf
        mov     ah,05h
        int     21h

        ret
CRLF    endp

;************************************************************
;
;    BLNK is a subroutine to output four blanks
;
;************************************************************

BLNK    proc    near

        mov     dx,20h          ;four blanks
        mov     ah,05h
        int     21h

        ret
BLNK    endp

;************************************************************
```

```
;
;    LST is a subroutine to list the contents of memory in
;a binary format;
;
;    Note: will only list the contents of memory stored in
;memory location LIST
;
;*****************************************************************


LST     proc    near

        mov     SI,offset LIST   ;SI points to LIST

;Calculate length of list to display

        mov     al,T             ;ax has number of errors
        cmp     al,02h           ;two errors?
        je      Two
        sub     cx,cx            ;clear cx
        mov     cl,N             ;n single errors
        jmp     Show

Two:    sub     cx,cx            ;clear cx
        sub     bx,bx            ;clear bx
        mov     cl,N             ;length of polynomial
        mov     bl,N             ;length of polynomial
        dec     bx               ;bx has N-1
Agn:    add     cx,bx            ;cx has # of error polys
        cmp     bx,000h          ;all done?
        je      Show             ;yes, print the results
        dec     bx               ;next number to add
        jmp     Agn

;Display the list

Show:   mov     bx,[SI]          ;polynomial to be
                                 ;displayed
        call    Prnt             ;print the results
        call    BLNK             ;add some blanks
        inc     SI
        inc     SI               ;point to next record
        mov     bx,[SI]          ;load bx with next record
        call    Prnt             ;print the next record
        call    CRLF             ;Cr and Lf
        inc     SI
        inc     SI               ;point to next record
        dec     cx               ;decrement counter
        cmp     cx,00h           ;all done?
        je      DN               ;yes, exit
        jmp     Show             ;no, get next record
```

```
DN:     ret
LST     endp
;*************************************************************

;All do, so exit from program
;
Exit:   mov     ax,4c00h    ;use DOS function 4c
        int     21h         ;to exit from program
;
main    endp
LAB_1   ends
        end     main        ;end assy
^Z
```

Appendix H: Demonstration Software

```
;*************************************************************
;
;     FDEMOENC.ASM
;
;     FDEMOENC.ASM is the software needed to run the final
;demonstration of the BCH encoder/decoder system.
;FDEMOENC.ASM stands for Final DEMO ENCoder.ASeMbly
;language, and is meant to run the encoder portion of the
;encoder/decoder system.
;
;     FDEMOENC.ASM will initialize the Z-80 DART to 9600
;baud, one stop bit, no parity, and x16 clock.  The user
;will be prompted for any ASCII character which will be sent
;to the encoder for multiplication by the generator
;polynomial.  The resulting code word is displayed on the
;screen, and the user prompted with the message "Do you wish
;to enter an error polynomial?".
;
;     If the user answers no, the code word is sent to the
;decoder without modification.  When the user answers yes,
;FDEMOENC.ASM will prompt the user for the error polynomial
;which is subsequently entered by the user into the
;computer. The error polynomial is exclusive-ored to the
;code word to  produce the transmitted code word.  The
;transmitted code word is finally sent to the decoder for
;decoding.
;
;*************************************************************
;
;  Define constants
;
bell            equ         07h     ;ASCII char for bell
cr              equ         0dh     ;ASCII char for carriage
                                    ;return
lf              equ         0ah     ;ASCII char for line feed
port1           equ         0000h   ;output port, data
port2           equ         0001h   ;output port, control
;
;
;*************************************************************
;
stack_area      segment     stack           ;define the stack area
                db          400 dup(?)      ;set aside room for
stack
stack_area      ends
;
;
data_area       segment                     ;define data area
```

```
msg1      db        cr,lf
          db      'Do you wish to enter an error
                    polynomial?','$'
          db        cr,lf,'$'
msg2      db        cr,lf,lf
          db        'Enter the code word:',cr,lf,'$'
msg3      db        'Please enter the error polynomial:','$'
msg4      db        'OK','$'
msg5      db        '    Encoded word is:  ','$'
msg6      db        'Error polynomial is:  ','$'
msg7      db        'Transmitted word is:  ','$'
GEN       dw        01d1h
CODE      dw        0000h
MWORD     dw        0000h
TWORD     db        00h
TRCD      dw        0000h
ERRP      dw        0000h
NK        db        08h
data_area      ends
;
;*****************************************************************
;
;
LAB_1          segment          ;define segment area for
program
main           proc       far
               assume     cs:LAB_1, ds:data_area,
                          ss:stack_area
;
;set all register to zero (makes it easier to debug)
;
begin:         sub        ax,ax                ;set ax to zero
               sub        bx,bx                ;set bx to zero
               sub        cx,cx                ;set cx to zero
               sub        dx,dx                ;set dx to zero
;
;set DS reg to point to the current data area
;
          mov        ax,data_area    ;point to data_area
          mov        ds,ax           ;DS now points to
                                     ;data_area

          Call       I_DART
Tloop:    Call       Get_WORD
          Call       Code_WORD
          Call       TErr
          Call       Send_WORD
          jmp        Tloop

;*****************************************************************
;
;   I_DART is a subroutine to initialize the DART to 9600
```

```
;baud, one stop bit, no parity, and x16 clock
;
;*****************************************************************

I_DART   proc      near
         push      ax
         push      bx
         push      cx
         push      dx

;
;  Initialize the DART to x16 clock, 1 stop bit, 8 Tx/Rx
;bits, no parity, and no interrupts (polled mode).
;
;       AL contains the control byte to the DART
;       DX contains the control byte address

           mov       dx,port2      ;output port address
           mov       al,18h        ;channel reset command
           out       dx,al         ;issue command
           out       dx,al         ;again, just to be sure
WR1:       mov       al,01h        ;point to WR1
           out       dx,al         ;load pointer into WR0
           mov       al,00h        ;disable all interrupts
           out       dx,al         ;issue command
WR2:       mov       al,02h        ;point to WR2
           out       dx,al         ;load pointer into WR0
           mov       al,00h        ;disable all interrupts
           out       dx,al         ;issue command
WR3:       mov       al,03h        ;point to WR3
           out       dx,al         ;load pointer into WR0
           mov       al,0c1h       ;Rx enable, 8 bits,
                                   ;auto enable off
           out       dx,al         ;issue command
WR4:       mov       al,04h        ;point to WR4
           out       dx,al         ;load pointer into WR0
           mov       al,44h        ;x16 clk, no parity, 1 stop
                                   ;bit
           out       dx,al         ;issue command
WR5:       mov       al,05h        ;point to WR5
           out       dx,al         ;load pointer into WR0
           mov       al,6ah        ;Tx enable, 8 Tx bits,RTS
                                   ;enable
           out       dx,al         ;issue command

;finished intializing, so restore registers

           pop       dx
           pop       cx
           pop       bx
           pop       ax
           ret
```

```
I_DART   endp

;*****************************************************************
;
;  Get_Word is a subroutine that gets the source word as
;a user input and stores that word in WORD.
;
;*****************************************************************

Get_WORD   proc    near

         push      ax
         push      bx
         push      cx
         push      dx

         mov       TWORD,00h        ;clear word buffer
         mov       dx,offset msg2   ;point to string
         mov       ah,09h           ;print string function
         int       21h

         mov       ah,01h           ;read char and echo
         int       21h
         mov       TWORD,al         ;store char in WORD

         call      CRLF

         pop       dx
         pop       cx
         pop       bx
         pop       ax
         ret
Get_WORD   endp

;*****************************************************************
;
;  Code_WORD is a subroutine that encodes the source word
;and stores the result in  MWORD
;
;*****************************************************************

Code_WORD  proc near

           push     ax
           push     bx
           push     cx
           push     dx
;
;  Disable the DART receiver and transmitter, so that
;  the BCH encoder can be used without affecting ;
transmissions.
;
```

```
        mov     dx,port2        ;control port
        mov     al,03h          ;point to WR3
        out     dx,al           ;load pointer into WR0
        mov     al,00h          ;Rx disabled, 8 bits,
        out     dx,al           ;auto enable off

        mov     al,05h          ;point to WR5
        out     dx,al           ;load pointer into WR0
        mov     al,00h          ;Tx disabled, 8 Tx bits,
        out     dx,al           ;RTS disabled

;
;   Generate the encoded word using the BCH encoder
;
        mov     MWORD,0000h     ;clear word buffer
        sub     bx,bx           ;clear bx
        mov     bl,TWORD        ;bx has code word
        mov     cl,NK           ;cl loaded with n-k
        inc     cl              ;one more shift req
        shl     bx,cl           ;add zero padding
        mov     CODE,bx         ;store padded code

        mov     dx,8000h        ;to clear the encoder
        mov     al,00h
        out     dx,al

        mov     cx,000fh        ;cx counts 15 bits
CLOOP:  mov     bx,CODE         ;bx has code char
        mov     dx,4000h        ;to clock data into
                                ;BCH encoder
        mov     al,bh           ;mov bh into al
        out     dx,al           ;output high bit
        shl     bx,1            ;shift bx left 1 bit
        mov     CODE,bx         ;store code word

;   Get the encoded word

        mov     dx,0c000h       ;to input a bit
        sub     bx,bx           ;clear bx
        in      al,dx           ;al has low order bit
        and     al,01h          ;strip off low bit
        mov     bl,al           ;bl has low order bit
        mov     ax,MWORD        ;al has encoded word
        or      ax,bx           ;append new bit

        mov     MWORD,ax        ;store code word
        cmp     cx,0001h        ;all done?
        je      Cxit            ;yes, Print result

        shl     ax,1            ;shift al left 1 bit
        mov     MWORD,ax        ;store code word in
                                ;MWORD
```

```
        dec     cx              ;cx holds count
        jmp     CLOOP           ;finish getting result
;
Cxit:   mov     dx,offset msg5  ;encoded word msg
        mov     ah,09h
        int     21h

        mov     bx,MWORD        ;encoded word in bx
        call    Prnt
        pop     dx
        pop     cx
        pop     bx
        pop     ax
        ret
Code_WORD   endp


;*****************************************************************
;
;  Subroutine to print the result
;  Word to be printed must be in bx
;
;*****************************************************************

Prnt    proc    near
        push    ax
        push    bx
        push    cx
        push    dx

        mov     cx,0010h        ;cx counts 16 bits

Loop2:  mov     ax,bx           ;put answer in ax
        and     ax,8000h        ;mask for high order
                                ;bit
        cmp     ax,0000h        ;is bit a zero?
        je      Zero
        mov     dl,31h          ;output a "1"
        mov     ah,02h
        int     21h
        jmp     Cont

Zero:   mov     dl,30h          ;output a "0"
        mov     ah,02h
        int     21h

Cont:   dec     cx              ;decrease counter
        cmp     cx,0000h        ;all done?
        je      Go              ;yes, exit
        shl     bx,1            ;get next bit and
        jmp     Loop2           ;continue
```

```
Go:       call    CRLF
          pop       dx
          pop       cx
          pop       bx
          pop       ax
          ret
Prnt      endp

;*****************************************************************
;
;     CRLF is a subroutine to print a CR and LF
;
;*****************************************************************

CRLF      proc    near
          push    ax
          push    bx
          push    cx
          push    dx

          mov     ah,02h          ;print character
          mov     dl,0dh          ;CR
          int     21h

          mov     dl,0ah          ;LF
          mov     ah,02h
          int     21h

          pop     dx
          pop     cx
          pop     bx
          pop     ax
          ret
CRLF        endp

;********
;subroutine to check

CK    proc    near
      push    ax
      push    bx
      push    cx
      push    dx

      mov     dx,offset msg4
      mov     ah,09h
      int     21h

      pop     dx
      pop     cx
      pop     bx
      pop     ax
```

```
        ret
CK      endp

;*****************************************************************
;
;   TErr is a subroutine that gets the error polynomial from
;the user and exclusive-ors it to the code word.  The code
;word is stored in MWORD
;
;*****************************************************************

TErr  proc   near

        push    ax
        push    bx
        push    cx
        push    dx

;  Prompt for user intent

        mov     dx,offset msg1      ;does user want an error
                                    ;poly?
        mov     ah,09h              ;print string call
        int     21h

        mov     ah,01h              ;get character and echo
        int     21h
        cmp     al,79h              ;y typed?
        je      Rever               ;yes, get error poly
        cmp     al,59h              ;Y typed?
        je      Rever               ;yes, get error poly
        mov     ax,MWORD            ;encoded word is TRCD
        mov     TRCD,ax
        jmp     Edone               ;no exit

Rever:  call    CRLF
        mov     dx,offset msg3      ;"Enter error poly"
        mov     ah,09h              ;print string call
        int     21h
        call    CRLF

;  Get error poly and process

        sub     bx,bx               ;clear bx
JC:     mov     ah,01h              ;read keyboard and echo
        int     21h

        cmp     al,30h              ;is it a zero?
        je      JZero
        cmp     al,31h              ;is it a one?
        je      One
```

```
        cmp     al,0dh              ;is it a CR
        je      JXor
        jmp     JC

JZero:  shl     bx,1                ;put 0 in bx
        jmp     JC
One:    mov     ax,0001             ;to add one to bx
        shl     bx,1                ;position poly
        or      bx,ax               ;add one to bx
        jmp     JC

JXor:   mov     ERRP,bx             ;store error in bx
        mov     dx,MWORD            ;code poly in dx
        xor     dx,bx               ;xor error poly
        mov     TRCD,dx             ;store trans poly
        mov     bx,dx               ;trans poly in bx
        call    CRLF
        call    Prnt

Edone:   call    CRLF
         pop     dx
         pop     cx
         pop     bx
         pop     ax
         ret
TErr       endp

;*****************************************************************
;
;    Send_WORD will send the final code word to the decoder
;    Note that Send_WORD waits for a request prior to sending
;    the code word.
;
;*****************************************************************

Send_WORD  proc    near
           push    ax
           push    bx
           push    cx
           push    dx

;  Display encoded word, error polynomial, and transmitted
;  word on the screen

           mov     dx,offset msg5
           mov     ah,09h
           int     21h

           mov     bx,MWORD
           call    Prnt

           mov     dx,offset msg6
```

```
            mov     ah,09h
            int     21h

            mov     bx,ERRP
            call    Prnt

            mov     dx,offset msg7
            mov     ah,09h
            int     21h

            mov     bx,TRCD
            call    Prnt

;   Enable DART transmissions

            mov     dx,port2            ;control port
            mov     al,03h              ;point to WR3
            out     dx,al               ;load pointer into WR0
            mov     al,0c1h             ;Rx enabled, 8 bits
            out     dx,al               ;autoenable off

            mov     al,05h              ;point to WR5
            out     dx,al               ;load pointer into WR0
            mov     al,6ah              ;Tx enabled, 8 tx bits
            out     dx,al               ;RTS enabled

;   Get a request to send from the decoder

Soop1:      mov     dx,01h              ;control port
            in      al,dx               ;rec buffer full?
            and     al,01h              ;mask for 1st bit
            cmp     al,00h              ;rec buffer full?
            jz      Soop1               ;no, wait for word
            mov     dx,00h              ;data port of DART
            in      al,dx               ;word in al
            cmp     al,52h              ;capital R sent?
            je      Sext1               ;go send code word
            jmp     Soop1               ;no, wait for R

;   Send the code word

Sext1:      mov     bx,TRCD             ;code word in bx
            mov     ah,04h              ;Aux output call
            mov     dl,bh               ;high byte first
            int     21h

            mov     dl,bl               ;low byte next
            int     21h

;   Clear buffers

            mov     MWORD,0000h
```

```
            mov     ERRP,0000h
            mov     TRCD,0000h

             pop     dx
             pop     cx
             pop     bx
             pop     ax
             ret
Send_WORD       endp
;
;*****************************************************************

;All do, so exit from program
;
Exit:     mov       ax,4c00h     ;use DOS function 4c
          int       21h          ;to exit from program
;
main      endp
LAB_1     ends
          end       main         ;end assy
^Z


;*****************************************************************
;
;   FDEMODEC.ASM is the software for the final demonstration
;of the decoder.
;
;   FDEMODEC.ASM begins by building a table for all the
;sydromes for the (15,7) BCH two error correcting code using
;the generator polynomial x^8 + x^7 + x^6 + x^4 + 1.  Then
;it initializes the Z-80 DART to 9600 baud, x16, one stop
;bit and no parity.
;
;   FDEMODEC.ASM will accept a 16 bit binay bit stream as
;the received encoded word.  An ASCII "R" (symbolizing ready
;to receive) is outputted through the auxilliary port to the
;transmitter.  After the transmitter has received the "R",
;the transmitter sends the transmitted word via its
;auxilliary port to the decoders Z-80 DART.  The received
;encoded word is sent to the decoder for decoding.
;
;   If the transmitted word had no errors, the transmitted
;word and the final result (ASCII character) are displayed
;on the screen for the user.  If the transmitted word has
;one or two errors, the transmitted word, syndrome
;polynomial, corrected transmitted word, and final result
;are displayed on the screen for the user.  In the advent
;that three or more errors are transmitted in the encoded
;word and the decoder is unable to resolve the error, it
;will display an uncorrectable error message.
;
```

```
;*****************************************************************
;
;  Define constants
;
bell            equ         07h     ;ASCII char for bell
cr              equ         0dh     ;ASCII char for carriage
                                    ;return
lf              equ         0ah     ;ASCII char for line feed
port1           equ         0000h   ;output port, data
port2           equ         0001h   ;output port, control
;
;*****************************************************************
;
stack_area      segment     stack           ;define the stack area
                db          80 dup(?)       ;set aside room for
stack
stack_area      ends
;
;
data_area       segment                     ;define data area
msg9      db        'Uncorrectable error transmitted','$'
msg1      db        'Transmitted word is:','$'
msg2      db        'Syndrome is:        ','$'
msg3      db        'Error polynomial is:','$'
msg4      db        'Correct code word:  ','$'
msg5      db        'Final Result is:    ','$'
GEN       dw        01d1h
CODE      dw        0000h
MWORD     dw        0000h
NK        db        08h
K         db        07h
N         db        0fh
T         db        02h
Base      dw        0001h
Drv       dw        0002h
RESULT    db        00h
MSK       dw        0000h
MSYND     dw        0000h
TRCD      dw        0000h
Count     db        00h
LIST      db    300 dup ('     ')
Extr      db     10 dup ('     ')
data_area       ends
;
;*****************************************************************
;
;
LAB_1           segment         ;define segment area for
program
main            proc        far
                assume      cs:LAB_1, ds:data_area,
                            ss:stack_area
```

```
;
;set all register to zero (makes it easier to debug)
;
                sub       ax,ax                ;set ax to zero
                sub       bx,bx                ;set bx to zero
                sub       cx,cx                ;set cx to zero
                sub       dx,dx                ;set dx to zero
;
;set DS reg to point to the current data area
;
          mov       ax,data_area    ;point to data_area
          mov       ds,ax           ;DS now points to
                                    ;data_area

          call      I_DART
          call      Generate
;         call      LST
MyLp:     call      Get_WORD
          call      Dec_Code
          jmp       MyLp


;****************************************************************
;
;   I_DART is a subroutine to initialize the DART to 9600
;baud, one stop bit, no parity, and x16 clock
;
;****************************************************************

I_DART    proc      near
          push      ax
          push      bx
          push      cx
          push      dx

;
;  Initialize the DART to x16 clock, 1 stop bit, 8 Tx/Rx ;
bits, no parity, and no interrupts (polled mode).
;
;       AL contains the control byte to the DART
;       DX contains the control byte address

            mov       dx,port2      ;output port address
            mov       al,18h        ;channel reset command
            out       dx,al         ;issue command
            out       dx,al         ;again, just to be sure
WR1:        mov       al,01h        ;point to WR1
            out       dx,al         ;load pointer into WR0
            mov       al,00h        ;disable all interrupts
            out       dx,al         ;issue command
WR2:        mov       al,02h        ;point to WR2
            out       dx,al         ;load pointer into WR0
```

```
        mov     al,00h          ;disable all interrupts
        out     dx,al           ;issue command
WR3:    mov     al,03h          ;point to WR3
        out     dx,al           ;load pointer into WR0
        mov     al,0c1h         ;Rx enable, 8 bits,
                                ;auto enable off
        out     dx,al           ;issue command
WR4:    mov     al,04h          ;point to WR4
        out     dx,al           ;load pointer into WR0
        mov     al,44h          ;x16 clk, no parity, 1 stop
                                ;bit
        out     dx,al           ;issue command
WR5:    mov     al,05h          ;point to WR5
        out     dx,al           ;load pointer into WR0
        mov     al,6ah          ;Tx enable, 8 Tx bits,RTS
                                ;enable
        out     dx,al           ;issue command

;finished intializing, so restore registers

        pop     dx
        pop     cx
        pop     bx
        pop     ax
        ret
I_DART  endp

;*****************************************************************
;
;   Get_WORD will get the final code word for the decoder
;
;*****************************************************************

Get_WORD  proc   near
          push   ax
          push   bx
          push   cx
          push   dx

;  Output a ready to receive (ASCII "R")

Glp:      mov    dl,52h           ;ASCII "R"
          mov    ah,04h           ;AUX output call
          int    21h

;  Enable DART transmissions

          mov    dx,port2         ;control port
          mov    al,03h           ;point to WR3
          out    dx,al            ;load pointer into WR0
          mov    al,0c1h          ;Rx enabled, 8 bits
          out    dx,al            ;autoenable off
```

```
            mov     al,05h          ;point to WR5
            out     dx,al           ;load pointer into WR0
            mov     al,6ah          ;Tx enabled, 8 tx bits
            out     dx,al           ;RTS enabled

;   Check input buffer:

            mov     dx,01h          ;control port
            in      al,dx           ;rec buffer full?
            and     al,01h          ;mask for 1st bit
            cmp     al,00h          ;rec buffer full?
            jz      Glp             ;no, send ready again

;   Get high byte

            mov     dx,00h          ;data port of DART
            in      al,dx           ;word in al
            mov     bh,al           ;high bits in bh

;   Wait for low byte

GGlp:       mov     dx,01h          ;control port
            in      al,dx           ;rec buffer full?
            and     al,01h          ;mask for 1st bit
            cmp     al,00h          ;rec buffer full?
            jz      GGlp            ;wait for word

;   Get low byte

            mov     dx,00h          ;data port of DART
            in      al,dx           ;word in al
            mov     bl,al           ;low bits in bl
            mov     TRCD,bx         ;store transmitted word

;   Disable DART transmissions:

            mov     dx,port2        ;control port
            mov     al,03h          ;point to WR3
            out     dx,al           ;load pointer into WR0
            mov     al,00h          ;Rx disabled
            out     dx,al           ;Issue command

            mov     al,05h          ;point to WR5
            out     dx,al           ;load pointer into WR0
            mov     al,00h          ;Tx disabled
            out     dx,al           ;Issue command

             pop     dx
             pop     cx
             pop     bx
             pop     ax
```

```
            ret
Get_WORD      endp
;
;
;*****************************************************************
;
;   Generate is a subroutine that generates the syndroms for
;all possible error conditions allowed by the given code
;structure.
;
;   Note that cx contains the length of the encoded word
;         and dx contains the number of correctable errors
;
;*****************************************************************
;
Generate   proc   near
;
;  Generate single error polynomials
;
        push      ax
        push      bx
        push      cx
        push      dx
;
;  Clear LIST to all Zeros
;
        sub       bx,bx             ;bx has all zeros
        mov       SI,offset LIST    ;SI points to LIST
        mov       cx,300d           ;cx has number of
                                    ;records

Rpt:    mov       [SI],bx           ;output zeros
        add       SI,02h            ;point to next record
        mov       [SI],bx           ;output zeros
        add       SI,02h            ;point to next record
        dec       cx
        cmp       cx,00h            ;all done?
        je        Sta
        jmp       Rpt

;  Compute single errors:
;
Sta:    mov       SI,offset LIST    ;SI points to LIST
        mov       bx,0001h          ;bx used to generate
                                    ;error
                                    ;polynomials
        mov       cl,N              ;cx has length of
                                    ;polynomial
Goop1:  mov       [SI],bx           ;store bx in LIST
        call      DECODE            ;calculate the syndrom
        add       SI,02h            ;SI points to next pos
        mov       ax,MSYND
```

```
        mov     [SI],ax         ;store syndrome
        add     SI,02h          ;point to next position
        shl     bx,1            ;generate next error
                                ;poly
        dec     cl              ;count down number of
                                ;errors
        cmp     cl,01h          ;all done?
        je      Gext            ;do double errors
        jmp     Goop1           ;no, finish single
                                ;errors

; Compute double errors if number of errors = 2
;
Gext:   mov     cl,T            ;number of errors in cl
        cmp     cl,02h          ;double errors
                                ;indicated?
        jne     Gxit            ;no, skip double errors

        mov     cl,N            ;cx has length of
                                ;polynomial
        mov     dl,N            ;dx has length of
                                ;polynmoial
        dec     dl              ;dx has N-1
        mov     Count,dl        ;store N-1 in Count
        mov     bx,Base         ;bx used to generate
                                ;error
                                ;plynomials
        mov     ax,drv          ;ax has 02h

Goop2:  xor     bx,ax           ;bx contains double
                                ;error
        mov     [SI],bx         ;store bx in LIST
        call    DECODE          ;calculate the syndrome
        add     SI,02h          ;SI points to next pos
        mov     bx,MSYND
        mov     [SI],bx         ;store syndrome in LIST
        add     SI,02h          ;inc SI
        mov     bx,Base         ;reinitialize bx
        shl     ax,1            ;shift ax left once
        dec     dl              ;to count shifts
        cmp     dl,00h          ;all done yet
        je      Chang           ;yes change bx and ax
        jmp     Goop2           ;no continue

Chang:  dec     cl              ;decrease length by one
        cmp     cl,01h          ;all combinations done
                                ;yet?
        je      Gxit            ;all done, exit
        mov     bx,Base         ;load bx with Base
        shl     bx,1            ;shift base left once
        mov     Base,bx         ;store bx in Base
        mov     ax,Drv          ;ax has driver
```

```
        shl     ax,1            ;shift ax left once
        mov     Drv,ax          ;store new driver in Drv

        mov     dl,Count        ;put Count in dl
        dec     dl
        mov     Count,dl        ;store Count-1 in Count
        jmp     Goop2
Gxit:   pop     dx
        pop     cx
        pop     bx
        pop     ax

        ret
Generate    endp

;*****************************************************************
;
;   DECODE will find the syndrom for each error polynmial
;using the BCH decoder.
;
;   The error polynomial must be in bx when DECODE is
;called. SI points to LIST (initialized by the calling
;routine)
;
;*****************************************************************

DECODE  proc    near

        push    ax
        push    bx
        push    cx
        push    dx

        mov     dx,0000h        ;to clear the decoder
        mov     al,00h
        out     dx,al
;
;   Clock in n-k data bits
;
BHigh:  mov     al,NK           ;al has n-k
        mov     dl,K            ;dl has k
        add     al,dl           ;al has n
        mov     cl,16d          ;cl has 16
        sub     cl,al           ;cl has # of zeros
        shl     bx,cl           ;shift out zeros

        sub     cx,cx           ;clear cx
        mov     cl,NK           ;cx counts n-k bits
        mov     dx,4000h        ;to clock data into
                                ;BCH decoder

DOOP3:  mov     al,bh           ;high order bit in A7
```

```
          out       dx,al           ;output high bit
          shl       bx,1            ;shift bx left once
          dec       cx              ;count down n-k
          cmp       cx,0000h        ;all done?
          je        DEC             ;yes, finish decoding
          jmp       DOOP3           ;finish loading bits

;
;  Next k bits are the decoded word
;

DEC:      mov       cl,K            ;to count k bits
          mov       RESULT,00h      ;clear result
DOOP4:    mov       dx,0c000h       ;input a bit into CPU
          in        al,dx           ;bit is in A0
          and       al,01h          ;mask for D0 (A0)
          mov       dl,RESULT       ;decoder word in dl
          shl       dl,1            ;shift deocded word
                                    ;left 1
          xor       dl,al           ;add bit to decoded
                                    ;word
          mov       RESULT,dl       ;store result

          dec       cl              ;decrease count
          cmp       cl,00h          ;all done?
          je        SYND            ;yes, get syndrome

          mov       dx,4000h        ;to output a bit
          mov       al,bh           ;high order bit in A7
          out       dx,al           ;output the bit
          shl       bx,1            ;shift encoded word
                                    ;left 1
          jmp       DOOP4           ;finish decoding

;  Syndrome (remainder) is in the n-k registers forming
;the generator polynmial

SYND:     mov       dx,8000h        ;output a bit, gate
                                    ;open
          mov       al,bh           ;to output last bit
          mov       cl,08h          ;to count NK bits

;  One more time with gate closed.
          mov       dx,4000h        ;output, gate open
          out       dx,al           ;clock in last bit
          mov       al,00h          ;output zeros now
          sub       bx,bx           ;make sure bx clear

;  Get syndrome bits and continue to clock in bits with the
;gate open.

DOOP:     mov       dx,0c000h       ;to input a bit
```

```
            in          al,dx           ;bit in A0
            and         al,01h          ;mask for low bit
            xor         bl,al           ;bx has syndrome
            dec         cl              ;decrease bit count
            cmp         cl,00h          ;all done?
            je          Dstr            ;store the syndrome
            shl         bx,1            ;ready for next bit
            mov         dx,8000h        ;output a bit, gate
                                        ;open
            mov         al,00h          ;to output a zero
            out         dx,al           ;output, gate closed
            jmp         DOOP            ;repeat

Dstr:       mov         MSYND,bx        ;store syndrome in SYND


;  Restore registers
            pop         dx
            pop         cx
            pop         bx
            pop         ax
            ret

DECODE      endp

;*****************************************************************
;
;  Subroutine to print the result
;  Word to be printed must be in bx
;
;*****************************************************************

Prnt        proc        near

            push        ax
            push        bx
            push        cx
            push        dx

            mov         cx,0010h        ;cx counts 16 bits

Loop2:      mov         ax,bx           ;put answer in ax
            and         ax,8000h        ;mask for high order
                                        ;bit
            cmp         ax,0000h        ;is bit a zero?
            je          Zero
            mov         dl,31h          ;output a 1
            mov         ah,02h
            int         21h
            jmp         Cont

Zero:       mov         dl,30h          ;output a 0
```

```
        mov     ah,02h
        int     21h

Cont:   dec     cx              ;decrease counter
        cmp     cx,0000h        ;all done?
        je      MGo             ;yes, exit
        shl     bx,1            ;get next bit and
        jmp     Loop2           ;continue

MGo:    call    CRLF
        pop     dx
        pop     cx
        pop     bx
        pop     ax

        ret
Prnt    endp

;*************************************************************
;
;   CRLF is a subroutine to output a carriage return
;and a line feed.
;
;*************************************************************

CRLF    Proc    near

        mov     dl,0dh      ;cr
        mov     ah,02h
        int     21h

        mov     dl,0ah      ;lf
        mov     ah,02h
        int     21h

        ret
CRLF    endp

;*************************************************************
;
;    BLNK is a subroutine to output four blanks
;
;*************************************************************

BLNK    proc    near

        mov     dx,20h          ;four blanks
        mov     ah,02h
        int     21h

        ret
BLNK    endp
```

```
;*****************************************************************
;
;    LST is a subroutine to list the contents of memory in
;a binary format;
;
;    Note: will only list the contents of memory stored in
;memory location LIST
;
;*****************************************************************


LST     proc    near

        mov     SI,offset LIST   ;SI points to LIST

;Calculate length of list to display

        mov     al,T             ;ax has number of errors
        cmp     al,02h           ;two errors?
        je      Two
        sub     cx,cx            ;clear cx
        mov     cl,N             ;n single errors
        jmp     Show
Two:    sub     cx,cx            ;clear cx
        sub     bx,bx            ;clear bx
        mov     cl,N             ;length of polynomial
        mov     bl,N             ;length of polynomial
        dec     bx               ;bx has N-1
Agn:    add     cx,bx            ;cx has # of error polys
        cmp     bx,000h          ;all done?
        je      Show             ;yes, print the results
        dec     bx               ;next number to add
        jmp     Agn

;Display the list

Show:   mov     bx,[SI]          ;polynomial to be
                                 ;displayed
        call    Prnt             ;print the results
        call    BLNK             ;add some blanks
        inc     SI
        inc     SI               ;point to next record
        mov     bx,[SI]          ;load bx with next record

        call    Prnt             ;print the next record
        call    CRLF             ;Cr and Lf
        inc     SI
        inc     SI               ;point to next record
        dec     cx               ;decrement counter
        cmp     cx,00h           ;all done?
        je      DN               ;yes, exit
```

```
        jmp     Show                ;no, get next record

DN:     ret
LST     endp

;*************************************************************
;
;   Dec_Code is a subroutine that decodes the received
;transmitted code word.  Note that the transmitted
;code word is in TRCD
;
;*************************************************************

Dec_Code  proc  near
          push  ax
          push  bx
          push  cx
          push  dx

          mov   bx,TRCD       ;transmitted word in bx
          mov   dx,offset msg1
          mov   ah,09h
          int   21h
          call  Prnt
          call  DECODE
          mov   bx,MSYND      ;but syndrome in bx
          cmp   bx,0000h      ;syndrome = 0?
          je    DCR           ;go print result
          mov   dx,offset msg2
          mov   ah,09h
          int   21h
          call  Prnt          ;print syndrome

;  Now search list for correct error polynomial

          mov   SI,offset LIST  ;point to LIST
          add   SI,02h          ;point to syndrome
          mov   dl,8ch          ;search list
Clop:     mov   ax,[SI]
          mov   bx,MSYND
          cmp   bx,ax           ;syndromes match?
          je    SynM
          add   SI,04h          ;get next record
          dec   dl
          cmp   dl,00h          ;all done
          je    Ermsg
          jmp   Clop

SynM:     mov   ax,SI
          sub   ax,0002h
          mov   SI,ax           ;point to error poly
          mov   bx,[SI]         ;load error poly
```

```
            mov   dx,offset msg3
            mov   ah,09h
            int   21h
            call  Prnt
            mov   dx,TRCD       ;transmitted word in TRCD
            xor   bx,dx         ;correct trans word in bx
            mov   dx,offset msg4
            mov   ah,09h
            int   21h
            call  Prnt
            call  DECODE
            mov   bx,MSYND      ;syndrome in bx
            cmp   bx,0000h      ;syndrome = 0?
            je    DCR              ;yes, print result

Ermsg:      mov   dx,offset msg9  ;error msg
            mov   ah,09h          ;print string
            int   21h
            call  CRLF
            jmp   NL

DCR:        mov   dx,offset msg5  ;Final Reuslt msg
            mov   ah,09h
            int   21h

            mov   dl,RESULT
            mov   ah,02h          ;display char function
            int   21h
            call  CRLF
            mov   dl,02h
            int   21h
            call  CRLF

NL:         pop   dx
            pop   cx
            pop   bx
            pop   ax
            ret
Dec_Code    endp

;*****************************************************************

;All do, so exit from program
;
Exit:     mov       ax,4c00h    ;use DOS function 4c
          int       21h         ;to exit from program
;
main      endp
LAB_1     ends
          end       main        ;end assy
^Z
```

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; Distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) AFIT/GE/ENG/88D-20	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION School of Engineering 6b. OFFICE SYMBOL (If applicable) AFIT/ENG	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State, and ZIP Code) Air Force Institute of Technology Wright-Patterson AFB, OH. 45433	7b. ADDRESS (City, State, and ZIP Code)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Foreign Technology Division 8b. OFFICE SYMBOL (If applicable) FTD/SDJC	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code) FTD Wright-Patterson AFB, OH. 45433	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.
11. TITLE (Include Security Classification) See box 19	
12. PERSONAL AUTHOR(S) Norman R. LeClair, Capt, USAF	
13a. TYPE OF REPORT MS Thesis 13b. TIME COVERED FROM _____ TO _____	14. DATE OF REPORT (Year, Month, Day) 1988 December 15. PAGE COUNT 166
16. SUPPLEMENTARY NOTATION	
17. COSATI CODES FIELD / GROUP / SUB-GROUP	18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number) BCH Encoder, BCH Decoder, Asynchronous Serial Interface, Error Detection and Correction

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Title: HARDWARE IMPLEMENTATION OF A BCH ENCODER, DECODER, AND INTERFACE

Thesis Advisor: Glenn E. Prescott, Capt, USAF
Professor of Electrical Engineering

12 Jan 1989

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☐ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL Glenn E. Prescott, Capt, USAF	22b. TELEPHONE (Include Area Code) (513) 255-3576 22c. OFFICE SYMBOL AFIT/ENG

DD Form 1473, JUN 86 *Previous editions are obsolete.*

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

Abstract

A BCH encoder and decoder are implemented in hardware with special emphasis given to the encoder and decoder interfaces. The pitfalls of using a 7474 D flip flop as the basis of building the interface is discussed. The advantages of using a UART for the interface are outlined and the circuit diagrams to implement the UART interface in hardware are provided.

Finite impulse response linear filters are chosen to implement both the BCH encoder and decoder. A basic theoretical understanding of the BCH encoder and decoder function is given. Design decisions made for the hardware implementation of the encoder and decoder are discussed, and schematics detailing the final hardware configurations are provided. Software to run and test all of the above is documented in the appendices.